CHAPTER 5

Working with the Registry

In This Chapter

Determining what the registry is used for and what it can do for you

Understanding how the registry is organized

Deciding what kind of information and settings the registry should contain

Learning to add, remove, and modify the registry entries you create

Adding security to your registry entries as needed

No matter which operating system you use or type of programs you create, the need to store program and user settings is a constant. Even simple things like the toolbars that the user prefers to see at startup are important to ensuring maximum user productivity and program ease of use. At one time, everyone relied on INI files to store program information like current user settings. Unfortunately, the INI solution only led to a big mess as everyone tried to stuff their settings in either WIN.INI or SYSTEM.INI. Some situations occurred in which one vendor would actually overwrite the settings required by someone else. In addition, a lost INI file could mean a broken application. Today, we no longer have those problems because of the Windows Registry.

Windows NT still provides the WIN.INI and SYSTEM.INI files for compatibility purposes. These two files are only used for storing the configuration settings of older applications like those created for Windows 3.*x*. Since using WIN.INI and SYSTEM.INI represent a risk for both the user and administrator, you'll normally want to upgrade these older applications as quickly as possible.

The Windows Registry is a free-form database that contains all of the stored settings required to make just about any part of Windows work. Each vendor has their own section of the registry to use. Since the registry is a form of database, looking for a particular value is relatively easy. Even manual searches aren't particularly difficult if you know how the registry is organized. In addition, the registry is loaded when Windows starts, which means that the information an application requires will be available in memory—greatly speeding the application initialization process.

If you haven't guessed already, the material in this chapter is going to help you understand the registry better. The first section provides an overview of the registry from the programmer's perspective. (This section won't provide an exhaustive commentary on the precise contents of the registry — an impossible job since the registry on any given machine will contain unique entries.) We'll look at how this free-form database is organized so that you can better understand how to access the information you need. You'll also learn that, in some situations, there are very specific places to put the information you need to store.

The second section of this chapter will help you understand what you need to store in the registry and where you'll normally want to store it. The registry is designed to provide organized data storage, so it's important to consider storage needs up front. You'll want to spend some time making sure you can get any settings you put into the registry, back out.

Once you have some theory under your belt, we can look at the various kinds of tasks you can perform with the registry. What you'll see is a set of actions that very nearly match what you would do with any database (though we'll stop short of using an actual database access technology like ADO or ODBC). You'll learn how to add, modify, and remove information in the registry. Of course, mixed in with all of these tasks is the task of reading the registry and understanding what it has to say.

An Overview of the Registry

As previously mentioned, the registry is a free-form hierarchical database that allows anyone who needs to store application to do so in an organized fashion. Of course, the database structure implies some type of organization, which starts at the registry root and works its way into functional areas. The functional areas of the registry are organized into six hives as shown in Figure 5-1.

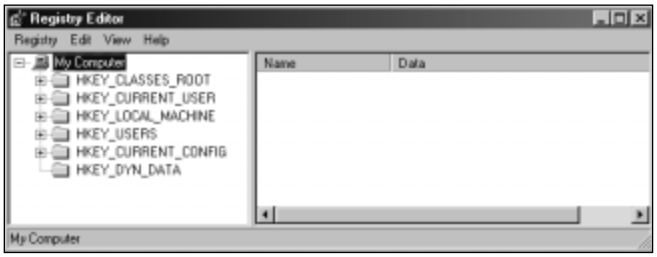

Figure 5-1: The registry contains six main functional storage areas designed to hold specific kinds of information.

Microsoft provides a utility for working with the registry named RegEdit (or Registry Editor). You'll find RegEdit.EXE in the main Windows folder. Even though this utility is always installed with Windows, it isn't linked to the Start menu so that novice users won't use what they don't see. The Registry Editor utility is a powerful program that you can use to create new registry entries as well as modify or view existing ones. Use this utility with care, since deleting a needed entry can cause your machine to freeze during boot or render it totally useless. Fortunately, Windows 2000, unlike Windows 95/98 products, protects many vital registry entries, making it harder to damage the registry beyond use. However, it's still a very good idea to make a backup of your machine before you perform much experimentation, and then proceed with caution.

Windows NT stores the current registry settings for each user in the NTUSER.DAT file found in each user's profile folder. The Default User folder contains the standard registry entries that Windows NT uses to create a new user. Since each user has a personal copy of the registry, any mistakes you make while editing the registry will affect only that user.

There are several ways to recover from an editing error. You can simply reverse the mistake while still in the registry. If you make a mistake and can't access the user's account when you try to enter it from the Log On dialog, you can always log on as the administrator and copy the registry file for a similar user to the damaged user's profile folder. You can also delete the user and add them back in. This would copy the Default User's registry to the new profile folder. A third option would include restoring a copy of the damaged registry file from a backup.

Obviously, there's one user whose registry file you'll never want to edit. Since just about every form of registry access requires an administrator account, damaging this registry file is one way to lock yourself out of the system. With that in mind, I always make sure that every machine has two users with full administrator rights: The administrator and myself (or a designated representative).

Before we go much further, there are a few terms you need to understand. The Windows NT registry uses two types of entries to maintain its organization: keys and values. The following list tells you about each type.

Keys *A registry topic.* Think of a key as the heading that tells you what a particular section contains. Looking at all the keys provides an outline for a particular topic. The topics could range from the setup of each drive in the system to the file associations needed to configure the machine. The key at the top of the hierarchy usually contains generic information. Each subkey provides a little more detail about that particular topic. Keys always appear in the left pane of the RegEdit window.

Values *The definition of a registry topic.* Values describe the key in some way and are divided into a value name and the data associated with that name. Think of a value as the text that fills out the heading provided by a key. Values are like the text in this book, while keys are the headings that organize the text. A value can contain just about anything. For example, the value for a file association key can tell you which application Windows NT will start when you double-click on a file that has that extension in Windows Explorer. A value could tell you which interrupt and I/O port settings a piece of hardware uses as well. You'll find the value you need using the keys, but you'll find the actual information you need by reading the values. Values always appear in the right RegEdit pane.

Most applications need to store more than one type of value. You might need to store one value as a string, another as a number. There are three types of values supported by the registry: binary, string, and DWORD. Applications normally use DWORD or binary values to store data, rather than application settings. For example, you might find the score from your last game of FreeCell stored as a binary or DWORD value. String values provide a lot of information about the application and how it's configured. Device drivers usually use string values as well for interrupt and port information. The following list provides more information about the three value types.

Binary A string of word values. These values can be any length and are usually numeric. In fact, you'll use a binary value to store floating-point numbers since they can't be stored any other way. However, you could just as easily store an object or a string as a binary value. The important thing to remember is that a binary value is treated as machine code—it's stored as absolute values.

String One or more plain-text (ASCII) characters. Strings are normally used for application settings instead of data, and normally appear in human readable format to make it easy for the administrator to make repairs to the registry. Some common strings used by programmers include executable file and path information. In addition, all CLSID (class identifier) information is stored as strings.

DWORD A single double-word value of 64 bits. DWORD values normally hold integer values. You'll find this kind of value used for quantitative purposes. For example, the processor speed of your machine is stored as a DWORD value.

Now that we've gotten the preliminaries out of the way, let's talk about the contents of each hive in more detail. The following sections will help you understand what kind of information you'll find in each hive and how it relates to Windows as a whole.

HKEY_CLASSES_ROOT

There are three distinct types of key entries in the HKEY_CLASSES_ROOT hive, even though you may not be able to figure out which is which at first. The first key type is a file association. This includes all the three-letter extensions you've used for application documents, such as .DOC and .TXT. Windows NT uses these keys to differentiate one file type from another. It also uses them to associate that file type with a specific action. For example, even though you can't do anything with a file that uses the .DLL extension, it appears in this list because Windows NT needs to associate DLLs with an executable file type.

The second key entry type is file association information, including OLE 1 entries. The file extension entries normally associate a data file with an application or an executable file with a specific Windows NT function. Below the association key are entries for the menus you see when you right-click on an entry in the Explorer. It also contains keys that determine which type of icon to display and other parameters associated with a particular file type. There's also a pointer here to the OLE 2 CLSID entries for this particular file association.

The third key entry type contains OLE 2 entries organized by CLSID. This 128-bit number is actually the GUID (globally unique identifier) that got assigned to the object in question as a means for differentiating between object types. As a minimum, each GUID entry will include either a pointer to another GUID or the location of the in-process server (InProc32 key) for the object. Many object entries will include type library information and the program identifier along with a wealth of other information we'll discuss as the chapter progresses. Figure 5-2 shows the typical HKEY_CLASSES_ROOT organization. Note that this figure doesn't show the CLSID key, which contains all of the OLE 2 entries.

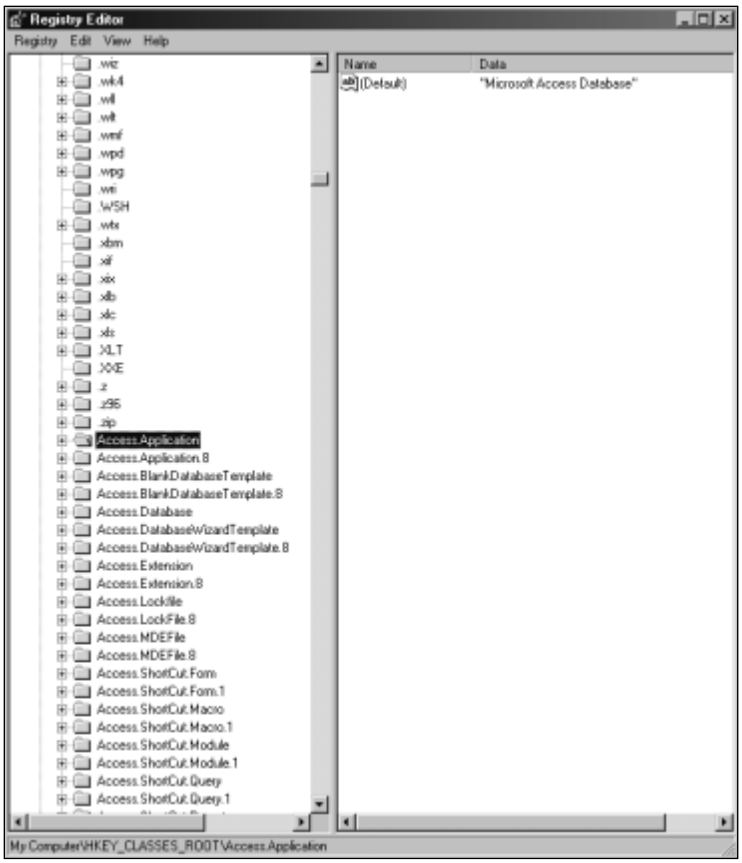

Figure 5-2: A typical HKEY_CLASSES_ROOT display. Notice the distinct difference between file extension and file association keys.

OLE 1 Registry Entries

Windows NT provides both OLE 1 and OLE 2 support, but in two entirely different sections of the HKEY_CLASSES_ROOT hive. Windows always places OLE 1 support information in the file association area. Figure 5-3 shows a typical file association hierarchy, but the actual setup varies between applications. Table 5-1 describes the various entries in this area (plus a few not supported by this particular application). The OLE support provided by each application on your machine will probably differ from the one shown in the figure; this is only an example for you to look at. Table 5-1 talks about all the common OLE 1 entries you might find. I don't think any application uses them all. (The table doesn't list many of the more esoteric entries, nor can it include any custom entries created by a specific application.)

Figure 5-3: Windows places OLE 1 support in the file association area of the registry.

Table 5-1
Common OLE 1 Entries in the Windows Registry

Entry	*Description*
AppRegistry	The location within the registry that contains the settings for the application associated with this file type. The string entry normally doesn't contain a complete registry path—it includes only the path needed to find the key within the HKEY_CURRENT_USER hive.
CLSID	This is actually part of the OLE 2 entry even though it appears in the OLE 1 section. It's a pointer to a GUID entry under the CLSID key. If you look for this number under the HKEY_CLASSES_ROOT | CLSID key, you'll find the OLE 2 entries for this file association.

Continued

Table 5-1 *(continued)*

Entry	***Description***
DefaultIcon	Contains a path to the icon used for this file type. The value for this key includes both the filename and the number of the icon within the file.
Insertable	Normally, you won't see any value associated with this key. A blank value means that Windows can place this file association in the Insert Object (or equivalent) dialog provided by many applications, such as Word for Windows. Some OLE 1 objects aren't insertable for a variety of reasons.
protocol	This is a header key (think of it as a heading in a book, a means to group like information together in one place). Underneath it you'll find all of the standard actions that this OLE 1 application can perform. The only supported function, in most cases, is a standard file edit.
StdFileEditing	This key is another header. In this case, it's the heading for all of the keys that will define a particular action—standard file editing.
server	You'll find the name of the application Windows will call to service any OLE 1 calls for this file association. The string always includes the application's name, extension, and path. If you ever run into a situation in which your OLE links worked yesterday, but won't work today, check this value to make certain the path matches the actual application location.
Verb	Several verbs are associated with an OLE 1 object. Each verb defines a specific action that the server will perform. This key doesn't have a value associated with it; its sole purpose is to organize actual verb entries. A client application can use only the verbs that are defined for a specific server. We'll talk about verb values in a little while.
RequestDataFormats	This entry enables the server to define what data formats it supports for retrieval purposes.
SetDataFormats	This entry enables the server to define what data formats it supports for storage purposes.
shell	This is a header key that contains a list of subkeys for actions that the user can perform with the file association from within Windows Explorer. These actions appear on the context menu that the user sees when they right-click the file. Each shell entry includes the name of the command and some type of command sequence for executing the request. In some cases, there is also a ddeexec key that tells how to perform a DDE (dynamic data exchange) sequence with the application in question.

The OLE1 entries also include a number of verbs — actions the user can perform on that particular file. The meaning of a particular verb entry depends on how the application reacts to that verb. In most cases, applications provide a standard set of verbs that they react to in a more or less predictable manner. Some applications may also define custom verbs that require some type of special action. Table 5-2 contains a list of the more common verbs and what they're used for.

Table 5-2
Common OLE1 Verb Definitions

Identifier	*Verb*	*Definition*
-3	Hide, 0, 1	This verb allows the client application to hide the server window. The first number following the verb is a menu flag. The second number is the verb flag. As a user, you don't need to worry about either value as long as you supply the values shown here (or use the settings provided by the vendor).
-2	Open, 0, 1	This verb allows the client to open the server in a separate window rather than have it to take over the client window.
-1	Show, 0, 1	A client would use this verb to display the server window in its preferred state. The whole idea of a state can get to be quite complex. Think of it as the way the window looks, and you'll have a pretty good idea of what to expect from a user's point of view.
0	&Edit, 0, 2	Every server provides this verb. It allows the client to call the server to edit the object.
1	&Play, 0, 3	The only time you'll see this verb is when you're looking at some form of multimedia object. It enables the client to perform whatever multimedia action is associated with the file.

OLE 2 Registry Entries

Now it's time to look at the OLE 2 part of the registry. You'll find these entries under the HKEY_CLASSES_ROOT | CLSID key. Each of the 128-digit numbers here represents a specific object GUID. Not every GUID you see here is associated with a document object. Some GUIDs are associated with ActiveX controls, and others are used for other purposes. In other words, you'll normally follow a three-step process when working with application documents.

1. Find the document extension in the upper part of the HKEY_CLASSES_ROOT key. Use the Default value of the document extension key to find the file association (OLE 1) entry in the lower part of HKEY_CLASSES_ROOT. For example, the .doc file extension on my machine points to the Word.Document.8 key.

2. Check the file association key for a CLSID entry. The Default value here will point to a particular OLE 2 GUID. For example, the CLSID value for the Word.Document.8 file association is {00020906-0000-0000-C000-000000000046}.
3. Use the GUID you obtained in Step 2 to find a particular entry under HKEY_CLASSES_ROOT | CLSID.

Once you find the GUID containing the entries for your document object, you'll see a hierarchical display similar to the one shown in Figure 5-4. Table 5-3 describes some of the more common entries that you see in detail. An application doesn't necessarily need to provide every entry that I describe in Table 5-4 in the registry. It depends on what OLE 2 features the application vendor decided to support. You can use these registry entries to determine the capabilities of your application, just like you can with OLE 1. (Not every key shown in Figure 5-4 is described in Table 5-3, since some of these keys are Microsoft-specific.)

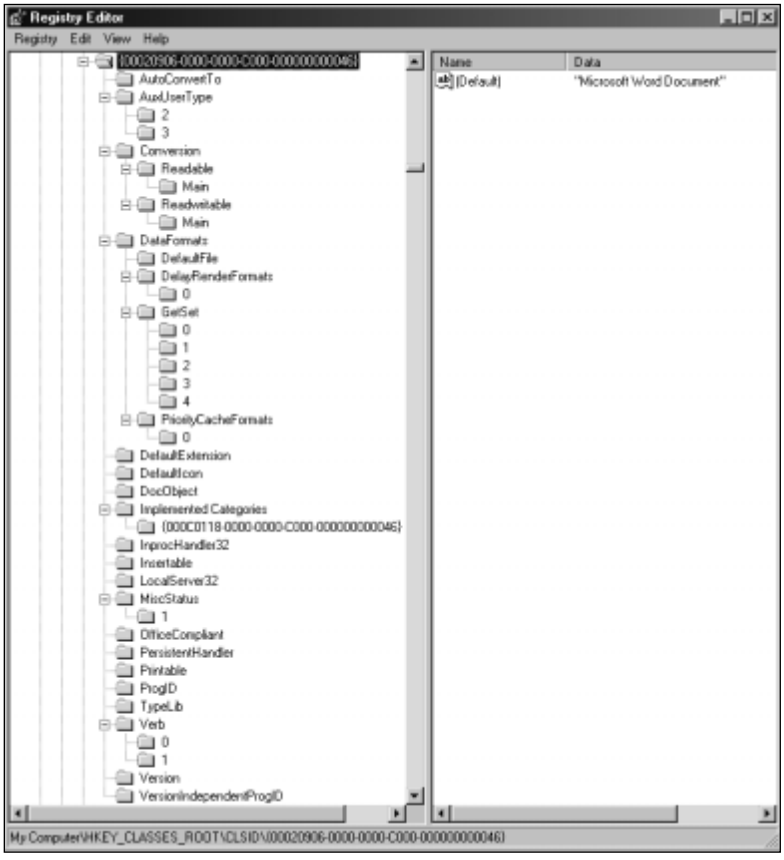

Figure 5-4: Windows places OLE 2 support in the file association area of the registry.

Table 5-3
Common OLE 2 Entries in the Windows Registry

Entry	***Description***
AutoConvertTo	Many applications define this key, but don't provide a default value for it. This key is supposed to provide Windows with the information it needs for automatically converting one file type to another. This comes in handy when the user doesn't have the application required to open the file installed on their machine.
AuxUserType	This key is a heading for all of the user type keys that follow.
2 (AuxUserType)	The 2 AuxUserType always contains the short name for the application. A client can use this name in a listbox to identify the object's owner.
3 (AuxUserType)	This key contains the full name of the application that created the object. Like the 2 key, a client could use the 3-key value to provide an English application name for the object.
DataFormats	This key is a heading for all of the data format keys that follow.
GetSet	This key is a subheading for all of the data formats the server can store and retrieve. For example, an OLE server like Word for Windows would support a DOC, RTF, and standard text format. Its OLE 2 entries would reflect this fact. Each entry below this one defines a specific format type.
n = format, aspect, medium, flag	Each of the sequentially numbered keys contains a different format. Format contains the type of format as a string. You might find some as easy to read here as "Rich Text Format" or as cryptic as "Embed_Source." In every case, this string tells you the name of the format that the application supports. The client displays this string (for get formats) in the dialog box where you select the format you want to use when performing a Paste Special command. Aspect tells the client which display orientation the object supports. This usually means portrait and/or landscape. You'll usually find a value of 1 here for portrait. Medium contains the supported format as a computer-readable number. Flag tells the client whether this is a get, set, or both format—a value of 1 is get, a value of 2 is set, and a value of 3 is both.
DefaultFile	This entry works much like GetSet, except it identifies the default file format for this particular object.

Continued

Table 8-1 *(continued)*

Entry	***Description***
DefaultIcon	The value of this key tells the client which application icon to use when displaying the object as an icon.
InProcHandler	This key contains the name of the in-process handler. In most cases, this is OLE 2.DLL unless the application provides its own OLE 2 handler. (A handler is a special program that helps two programs communicate.)
Insertable	Normally, you won't see any value associated with this key. A blank value means that Windows can place this file association in the Insert Object (or equivalent) dialog provided by many applications such as Word for Windows. Some OLE 1 objects aren't insertable for a variety of reasons.
LocalServer	Every OLE 2 object must have a server. This key contains the name of the server on the local machine. Since OLE 2 doesn't support RPCs (remote procedure calls), you'll always need a local server.
MiscStatus	This key contains the default value for all data format aspects.
ProgID	The program identifier is a pointer back to the file association that this class identifier belongs to. The file association is always a character string of some sort. It's the same string that you look for when you try to find the file association that goes with a file extension in the HKEY_CLASSES_ROOT hive. For example, if you looked at the ProgID value for Corel Chart, you would see CorelChart5. That's the same value we see in Figure 5-3 as the first key for the file association.
Verb	Several verbs are associated with an OLE 2 object. Each verb defines a specific action that the server will perform. This key doesn't have a value associated with it; its sole purpose is to organize actual verb entries. A client application can use only the verbs that are defined for a specific server. See Table 5-2 for a list of verbs and their meanings.
InProcServer	This is a special form of OLE server. Instead of calling the application that created the object, the client calls a DLL to handle any necessary display or editing functions. This has the advantage of speed; a DLL is faster than calling the executable. However, the programmer has to do a lot more coding to make this kind of interface work.

Entry	***Description***
TreatAs	When this key is present, it contains the CLSID for another file format. The client can treat the current file format like the specified file format. For example, if you looked at the Paintbrush Picture OLE 2 entry, you would find a TreatAs value with the same 128-digit value as a Bitmap Image. This tells you that Windows uses the same application to service Paintbrush Picture files as it does for Bitmap Image files. A little more research would tell you that the OLE 2 server for Paintbrush Pictures is Microsoft Paint.
AutoTreatAs	This key forces the client to treat the current file format the same way it would treat the file format specified by the CLSID. From a user perspective, it works just like the TreatAs entry that I described previously.
AutoConvert	Some objects' context menus contain a Convert option. This key allows you to automatically convert the current file format to the one identified by CLSID. For example, Word for Windows allows you to convert many types of data to a format that it specifically supports. This conversion process changes the embedded or linked object into a Word for Windows object. In other words, it changes the data into something that Word might have created itself.
Convertible	There are two levels of subkeys below this one. The first level contains two keys, Readable and Writeable. Below them are keys that contain a list of readable or writeable file formats that you can convert this file format to. For example, you'll find that Word for Windows supports several formats including Word Document. The number of entries in this area usually varies by the number of file filters you install for the application. For example, if you install Word Perfect file support for Word for Windows, then you'll likely find an entry for it here. Remember, though, that these are OLE 2 entries. Even if an application supports another application's file format as part of a Save As option, it still might not support it for OLE purposes.
Interfaces	This key contains a list of interfaces supported by the server. The value for this key will eventually contain the names of other ways of accessing the OLE server (other than from the local machine), but there aren't any applications that support it right now. For example, this entry could contain the types of network protocols that the application supports.

HKEY_CURRENT_USER

The HKEY_CURRENT_USER hive contains a lot of "soft" settings for the user currently logged onto the machine. These soft settings tell how to configure the Desktop and the keyboard. This hive also contains color settings and the configuration of the Start menu. All active user-specific settings appear in this hive.

It's important to understand that the HKEY_CURRENT_USER hive is slaved to the settings for the current user, this user who's logged into the machine at the time. This is differentiated from all the user configuration entries in the HKEY_USERS hive of the registry. The idea of a current user is important, especially when it comes to remote application installation or configuration, because the current user's rights affect the rights of your application. HKEY_CURRENT_USER is a dynamic setting category; the HKEY_USERS hive user-related keys in the registry contain static information. The registry copies the contents of the current user's entries from the HKEY_USERS hive into this category when they log into the operating system, and updates HKEY_USERS hive when they shut down or log off.

This is the area where Windows NT/2000 obtains new setting information and places any changes you make. As you can see from Figure 5-5, the keys within the HKEY_CURRENT_USER hive are pretty self-explanatory in most cases. All the entries adjust some type of user-specific setting, nothing that affects a global element such as a device driver. In other words, this particular hive is a scratch pad of sorts. It's very useful to remember this fact. If you make a mistake, you can always use the settings in HKEY_USERS to reset the values to their previous settings. The following sections describe each of the HKEY_CURRENT_USER keys in detail. (Only common keys appear in the following sections; some machines may have keys that are specific to that particular machine.)

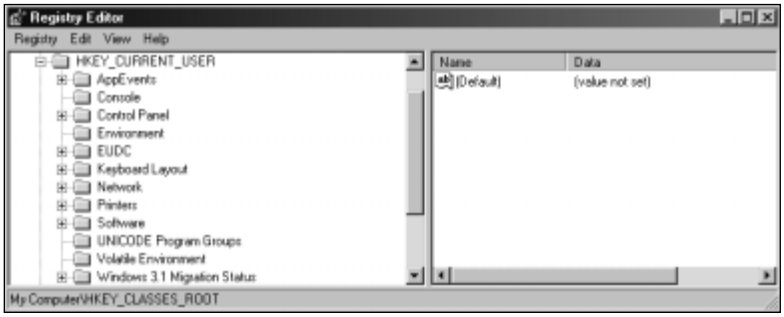

Figure 5-5: The HKEY_CURRENT_USER hive contains all user-specific settings.

AppEvents

Everyone likes to add a few sounds to their system to make it through the day. This key defines these sounds in a number of ways. Take a look at Figure 5-6 to see what the AppEvents subkeys look like. There are two subkeys under this one: EventLabels and Schemes. The EventLabels key contains a list of events. Each event key contains the long name of a sound event. For example, the EmptyRecycleBin event uses "Empty Recycle Bin" as a long name. This is what Windows displays in the dialog box you use to set the sound value.

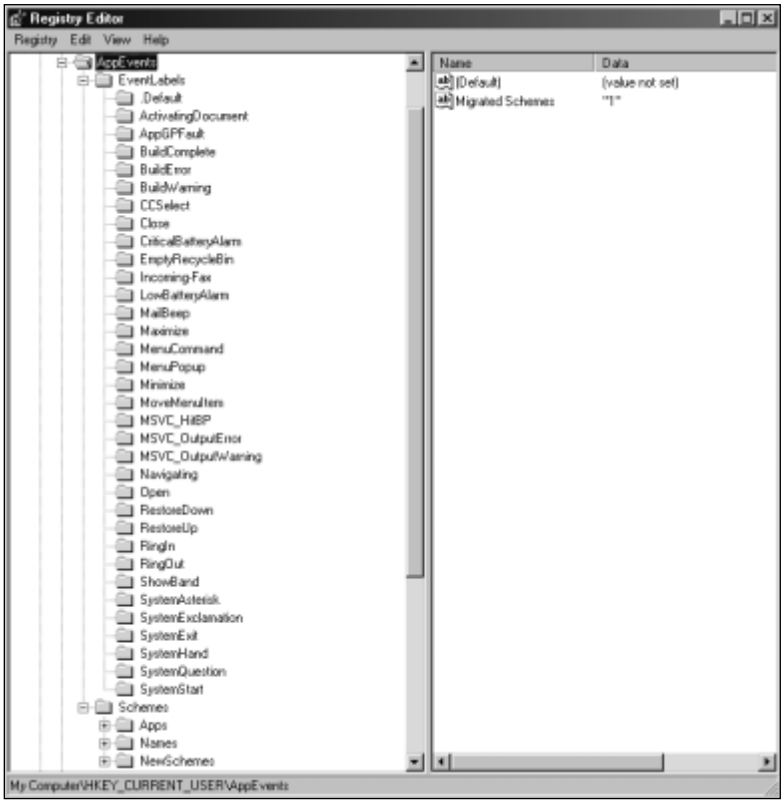

Figure 5-6: The AppEvents key controls the sounds for various system events and the strings used to display these event names in the Sounds Properties dialog.

The registry provides the only way you can change the strings displayed in the Sounds Properties dialog. You might want to use this option after changing the name of the Recycle Bin or other Windows components. In addition, you can use this key for adding special events to the Sound Properties dialog for your application. For example, many e-mail applications use this capability to create a special sound that signifies incoming mail.

The Schemes key is the one you'll probably want to explore. There are two subkeys below this one. The first one we'll look at is the Names key. The subkeys hold the full names of the various sound schemes you might have defined. The Apps key contains major subkeys for each kind of application that defines special sounds. The default sounds are the beeps and other general Windows noises you hear on a daily basis. If you looked at the values for each of the minor keys, you'd see the name of a sound file.

Console

Console applications require a DOS prompt to work. A DOS window requires settings to work, just like any other part of Windows NT. When you change the default DOS window settings, they appear in this area of the registry. Figure 5-7 shows a typical example of the Console key.

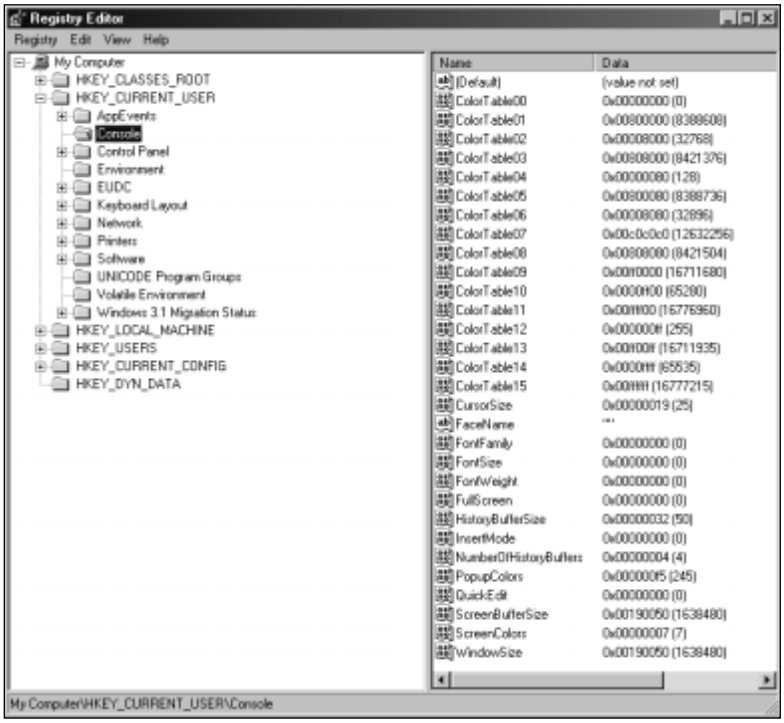

Figure 5-7: The Console key provides the default settings for a DOS window.

There isn't much potential here for modern applications unless you make use of a console application to run tasks in the background. However, there are some opportunities for customization. For example, you could change the color table entries to meet specific needs or change the FaceName value to make use of a better screen font.

Control Panel

This is probably the most familiar registry key. Figure 5-8 shows all of the subkeys that this key contains. As you can see, the Control Panel key doesn't contain subkeys for each icon in the Control Panel. However, it does contain keys for all of the icons that control user-specific settings. The following list describes these subkeys in more detail.

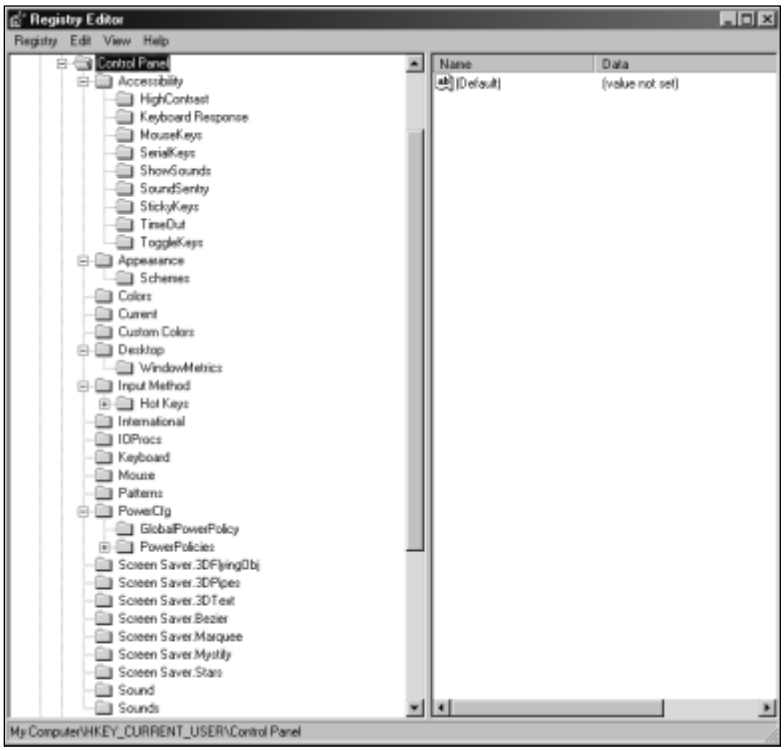

Figure 5-8: The Control Panel key contains subkeys for every user-specific feature on Control Panel.

Accessibility This subkey contains all of the settings for the accessibility features such as sticky keys and high-contrast monitor. You can find a complete description of these features in Chapter 4.

Appearance The Appearance key contains the current display's color settings. It includes both a text description and the custom colors. This value doesn't include the actual color values. Below this key is the Schemes subkey, which contains the text description and actual color value for all the color schemes you've defined.

Colors This subkey contains a list of object types and their associated colors. Windows uses these settings to present everything on your display. Theoretically, you could add entries to this key, but getting Windows to use them would be difficult at best.

The Color key settings provide a level of flexibility that you can't get using the standard Display Properties dialog. For example, clicking on the pushbutton displays the 3D object settings. These settings don't provide the same precise control as the Button values in this key. You could write code in your application to modify these values more precisely than the user could normally.

Current You'll find the current scheme setting here. This entry will tell you which scheme Windows is currently using for user settings. In most cases, you'll see a Color Scheme entry as a default — even if the user has selected the Windows Default scheme.

Cursors This subkey contains a list of cursor types and their associated cursor files. Below this key is the Schemes subkey. It contains a list of all the cursor schemes you have defined and their associated settings.

Custom Colors The values of any custom colors that you define appear here. This entry may or may not appear in the registry depending on the display adapter you're using and its capabilities.

Desktop All of your Desktop settings appear in the values of this key. It includes the current wallpaper, background, and screen-saver settings. Below this key is the ResourceLocale subkey, which contains the current country code. The WindowMetrics subkey defines the icon spacing and other graphic specifications for the Desktop.

International The International key contains the current country code.

IOProcs Leave this particular setting alone. It identifies the DLL that Windows NT uses for some types of input and output.

Keyboard This key contains your keyboard settings. It includes things like the amount of delay between key repeats and the speed at which Windows will accept input. You may not find this entry in some cases. Windows appears to add it if you modify the keyboard settings or use any of the keyboard-related accessibility features.

Mouse	This key contains the standard mouse settings, not the Accessibility mouse settings. It determines how the mouse reacts when you're using a standard setup.
Patterns	Windows uses this area to store all of the desktop patterns you define on the Background page of the Display Properties dialog.
ScreenSaver.XXXX	Each screen saver that you load as part of the Windows NT installation makes an entry in this area. It stores whatever settings it needs to display properly. In most cases, you'll find it easier to modify the screen-saver settings through the Display Properties dialog — there don't appear to be any "hidden" settings for the default screen savers provided with Windows. You may find some settings for non-Windows screen savers.
Sound	You'll find the sound settings here like extended sound and beep. This area doesn't contain the names of the WAV files associated with specific system events.
Sounds	The name of the sound scheme normally gets stored here.

Environment

This key contains the DOS environment settings. Windows stores them in hexadecimal instead of plain English. It uses these values when you start a DOS session. Windows applications can request this information as well.

InstallLocationsMRU

This registry key contains the most recently used locations of installation files. The entries you find here don't pertain to the location of software installation files, but to device or service-related installation files. For example, installing a new set of network drivers would place an entry here, but installing a new word processor wouldn't. Windows won't make this entry until you've installed some type of new device or service beyond the initial installation. Notice that there's a special value called MRUList. This value controls the order in which Windows displays the file names that appear in the list. If you do add an entry to the list, you also have to add it to the MRUList value.

Keyboard Layout

This registry key provides information about the keyboard currently attached to the machine. It doesn't really change from user to user unless one user needs to use a different language or keyboard layout. The Preload key under the Keyboard

section reflects the different languages in use. The Substitutes key reflects the keyboard layout. For example, you could use English (United States) as a language and English (Dvorak) as a layout. The Toggle key tells you which control-key combination you can use to switch between languages.

Network

This key provides user-specific network information. One key remains constant across networks — the Recent key. This key contains one subkey for each of the network drives you've recently accessed. Windows NT updates it only on a session basis, so you probably won't find current information here if you maintain your network connection for a long time between sessions. Each subkey uses the full network access path for a drive. For example, if you have a Microsoft network installed and you connect to the C drive of a machine named AUX, one of the subkeys might look like this: //AUX/DRIVE-C.

RemoteAccess

You may or may not find this key in your registry. It depends on which features you choose to install. Within the key itself is an area code value and a DWORD configuration value for the Make New Connection Wizard.

Below this key you'll find one subkey that holds the addresses you define. There's one binary value for each address you define. These values include all the configuration information for that address. Obviously, you won't want to edit this value outside the Dial-Up Networking application.

Software

This registry key should contain plenty of configuration information for your software. However, until you upgrade to 32-bit versions of your applications, it'll probably contain a few default entries for Windows 95 or Windows NT-specific applications. The SSC section contains information related to the viewers. The Microsoft section contains keys for a variety of applications, whether you have them installed or not.

The registry provides this section to replace all those entries you used to have in WIN.INI and SYSTEM.INI. The hierarchical structure of this key makes it very easy to remove application-specific information from your Windows NT installation.

Printers

This key contains the subkeys required to create a connection to network printers. The type of network printer the user creates a connection to influences the kind of information you see here. In most cases, you'll see individual settings for each network printer in the DevModePerUser key. You may also see a Connections key that may or may not contain any values. In other words, you'll need to research this key on the individual network before writing applications that read or manipulate it.

Software

This key contains a wealth of user settings for individual applications. There's no way to predict the exact contents of this key on a specific machine. However, this is the key you'll use for storing your application settings.

UNICODE Program Groups

Anyone who works with Windows NT for very long soon realizes that it supports UNICODE at the lowest level. What exactly is UNICODE? It's a method for representing characters in languages that use more than 26 characters. Each character uses 16 instead of 8 bits. This particular registry entry identifies program groups that use UNICODE entries instead of the standard 8-bit characters.

Volatile Environment

This area contains all of the current drive mappings — those that you made during the current session. The reason that it's called a volatile environment is that the settings will either become permanent or Windows will disregard them the next time you boot. Permanent settings get written to another area of the registry.

Windows 3.1 Migration Status

Windows NT provides methods for you to migrate from Windows 3.1. This registry entry tracks the status of that move. It contains a list of Windows elements that have migrated from the Windows 3.*x* environment to Windows NT. The two most common elements are INI files and group folders.

HKEY_LOCAL_MACHINE

The HKEY_LOCAL_MACHINE hive centers its attention on the machine hardware. This includes the drivers and configuration information required to run the hardware. Every piece of hardware appears somewhere in this section of the registry. If a piece of hardware doesn't appear here, Windows NT can't use it, even if the hardware is physically installed on your machine.

A lot of subtle information about your hardware is stored in this hive. For example, this hive contains all of the hardware configuration information about your machine. It also provides a complete listing of the device drivers and their revision level. This section might even contain the revision information for the hardware itself. For example, there's a distinct difference between a Pro Audio Spectrum 16+ Revision C sound board and a Revision D version of that same board. Windows NT stores that difference in the registry.

This hive does contain some software-specific information of a global nature. For example, a 32-bit application will store the location of its Setup and Format Table (SFT) here. This is a file that the application uses during installation. Some applications also use it during a setup modification. Applications, such as Word for Windows NT, store all their setup information in SFT tables. The only application information that does appear here is global-configuration-specific like the SFT. Figure 5-9 shows a typical HKEY_LOCAL_MACHINE hive setup. The following sections describe each key in detail.

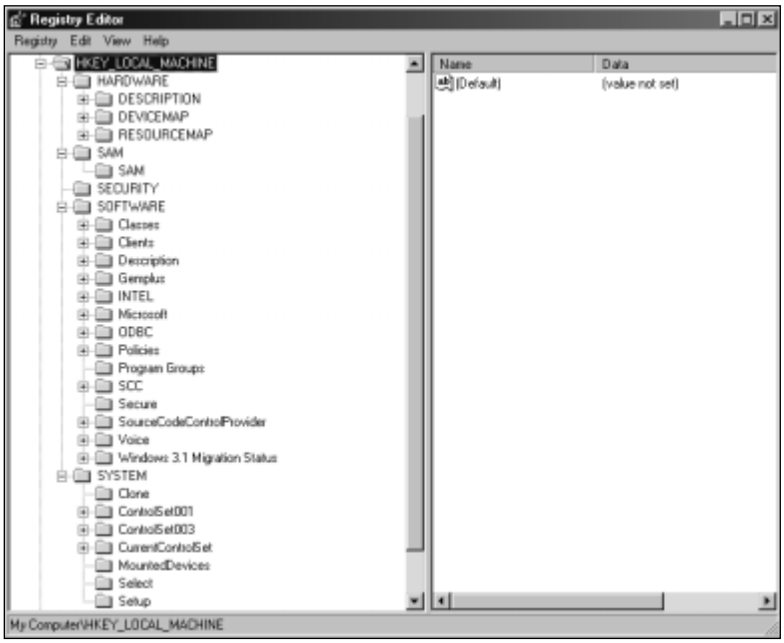

Figure 5-9: The HKEY_LOCAL_MACHINE hive contains the hardware, device-driver-specific, and global application setup information.

HARDWARE

Windows NT uses the HARDWARE key to store everything it needs to know about your hardware — the physical device, not the drivers that we'll discuss in a few moments. This includes everything from the processors that your machine uses to the devices attached to the disk-drive controllers. The following paragraphs describe the major entries in detail.

DESCRIPTION The primary function of the DESCRIPTION subkey is to list the communication ports, processor types and any floating-point processors attached to the machine. This key doesn't contain any actual configuration information — at least nothing that you would want to change.

DEVICEMAP	Windows NT places an entry here for each device in your machine that maps it in one of three ways to the system. The first entry type points to the device entry itself. In fact, it actually starts with the word "device" like this: "\Device\Video1". These are devices that don't require additional definition before Windows NT can use them. A second type of entry points to another place in the registry. It begins with the word registry like this: "\REGISTRY\Machine\System\ControlSet001\Service". You'll find this kind of entry attached to display adapters and mice, both of which require additional drivers to do their work. Finally, the third type of entry is SCSI-specific. It maps a controller to each drive in your system. The mapping can get quite complex and depends on what kind of adapter you're using, so I'm not going to get very specific here. The one interesting point about this particular entry is that you'll see it applied to some types of complex IDE controllers as well. Even though it's an IDE controller, Windows NT will still use the SCSI keyword.
OWNERMAP	You'll find a list of specialty owner drivers in this section. An owner driver is one that controls the action of other non-related devices. Most of them are bus-specific. For example, if you have a PCI bus, you'll find a driver for it here.
RESOURCEMAP	Every device on your system requires specific resources to work. The entries in this section have information about those resources. Most of the settings deal with the various device modes. For example, a serial port supports both raw and transmitted modes under Windows NT.

SECURITY

This registry key is network-specific. Refer to your network manuals for additional information. As a general rule, it contains pointers to files holding account and other network security-specific information. In some cases (usually with peer-to-peer networks), you won't see anything here, even if you have a network connected to your machine.

SOFTWARE

The SOFTWARE registry key contains device-driver and global-application information. Most applications store their specific configuration information under the Application key and any global information under another common key. Notice that the subkeys include a variety of vendor-specific information.

One of the interesting subkeys is Classes. Opening this key reveals a list of file extensions and associations just like the one in HKEY_CLASSES_ROOT. The purpose behind this set of keys is totally different, though. Windows NT uses this list to locate application- or device driver-specific files instead of creating links for Explorer or providing OLE configuration information. Changing any of these keys won't affect your Explorer display. Unfortunately, it might affect how some of your applications work. The bottom line is, don't change any of these settings unless you have a good reason to do so and know exactly how the change will affect your setup.

The Description key also provides some interesting information. A default setup always contains the Microsoft remote procedure call (RPC) listing. However, a network could use this section to describe other links between machines as well. An RPC allows your machine to use the resources found on another machine. This is an automatic feature. The application and operating system have to support RPCs. In addition, the resource has to be available — you can't steal someone's resource if they're already using it. RPCs can include all kinds of resources, but they normally refer to code. For example, you could borrow someone else's DLL if that DLL didn't appear on your machine. The unique aspect of RPCs is that the code for that DLL would execute on the other person's machine, not on your machine. You would, in effect, borrow not only their code, but their processor as well. Notice that this section doesn't define any light remote procedure call (LRPC) information. An LRPC is the same as an RPC except it deals with very specific types of resources and it always deals with code. You could borrow another person's DLL, but the code for that DLL would execute on your machine, not their machine. The LRPC statistics always appear as part of the OLE configuration information or within the RPC keys.

Unless you have a Microsoft mouse, there's a good chance that you'll have a mouse-specific entry — even if your mouse doesn't provide Windows NT-specific software. For example, installing a Logitech mouse adds a Logitech key that provides mouse-setup information as well as the driver version. Normally, the key provides values that determine which buttons become assigned to various mouse functions.

Using the registry to edit a hardware configuration setting is almost always more flexible than using the Control Panel settings. For example, what if you wanted to gain some additional control over your three-button Logitech mouse? The default setup program allows you to assign a left- or right-handed setup to the mouse. What if you wanted to exchange the right and middle button functions, essentially creating a two-button mouse? You could do so by transposing the MappingButton2 and MappingButton3 registry values even though the configuration utility doesn't provide the means to do so. Adding this capability to your application makes it possible for the user to create custom hardware setups that make using your application faster or easier.

The Microsoft section of the SOFTWARE key provides some useful information as well. One of these keys is the New Users Settings key. Most 32-bit applications and some 16-bit applications can provide an entry in this area. A new user inherits these settings when you set up a new login on the machine. In essence, this key contains the global settings that everyone uses with this application (at least until they customize their setup). This key also provides the location of shared tools and any global network settings required to make the application work.

If you install any applications that support open database connectivity (ODBC), those settings will appear under this key. ODBC is Microsoft's new method of allowing you to access information in foreign data formats. Its main purpose is to allow access to mainframe databases, but you'll find it used to access other sources of information including those commonly found on a PC. You'll also see the 32-bit ODBC settings here as well. The strange thing is that they aren't identified using separate keys, even though the 32-bit settings appear as separate icons in Control Panel. Every ODBC connection and its configuration appears under the ODBC | ODBCINST.INI key. The two important values are the name of the ODBC driver and its version. Most 32-bit drivers include a 32 somewhere in the filename. This fact allows you to separate the 32-bit drivers from the 16-bit drivers in this area.

The SCC key contains information about the viewers (if you installed them). At first glance, you might not think that there is anything worthwhile to look at. There are two keys at the bottom of the hierarchy, and both use descriptive value entry names such as Element 1. However, if you double-click on one of these entries, you'll find the name of the DLL that controls it embedded in the binary data that each value contains.

Under the Windows 3.1 Migration Status key are two entries: IniFiles and REG.DAT. The first key tells Windows NT whether it has taken all of the INI file data and placed it in the registry. The second key determines whether all the Windows 3.*x* registry data was incorporated into the registry.

SYSTEM

The last registry key in the HKEY_LOCAL_MACHINE hive is also the most destructive when modified incorrectly. A bad entry in any value under this key can kill your installation. In most cases, unless you're writing device drivers or other low-level software, it's best to leave this particular key alone.

The main purpose behind the SYSTEM key is device control. Complex devices require additional information to make them work. That's what you'll find in this subkey. Remember the REGISTRY subkey that I told you about that is part of the DEVICEMAP key? The pointer values in the REGISTRY key appear as keys here. Each one of those device maps appears in this section along with the current device map — the one that you're using right this second.

Obviously, there are other types of information here as well. If you look at one of the ControlSetXXX keys, you'll see a list of four subkeys: Control, Enum, Hardware Profiles, and Services. Let's take a look at the Control key first. It contains all of the subkeys that change the way the user reacts to the system as shown in Figure 5-10.

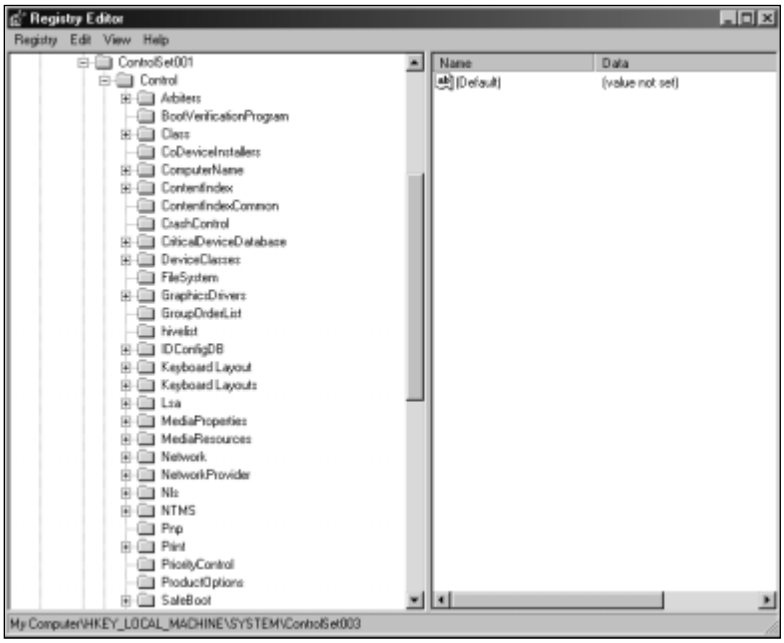

Figure 5-10: Each Control Set contains four subkeys, including the Control subkey shown here.

Exploring this part of the registry is somewhat risky. Some settings won't harm your system much—they might just change the text in a listbox or under an icon. However, other settings will damage your Windows NT installation beyond repair, and still others will prevent access to your system. Wandering through this section and seeing what'll happen when you change a setting is one sure way to make your system unusable. Always make a backup of your registry before you make any change to this section. In addition, it always pays to proceed with extreme caution—make only one change at a time and test it out thoroughly before making any other changes.

One of the keys you'll see here is the NLS (National Language Support) key. Remember the keyboard setting we looked at before? This is where all the language settings come from. If you had the correct driver, you could add support for Martian by changing the value of one of these keys. The network administration-specific keys appear here as well. One of the first keys you'll see is the computer name used to log onto the network.

Now let's look at the Enum key. Below it you may find an HTREE and a Root key. The HTREE key contains a ROOT subkey, under which you'll find a list of configuration keys. Each configuration key contains a binary value that tells you which pieces of equipment Windows NT has found attached to this system. In this case, this list is used for troubleshooting the system, not as part of an inventory. If you look at the Root key you'll see why. There's a list of devices below this key. Each device key also contains configuration keys. These configuration keys contain some human-readable information like the name of the device and the Windows NT service that supports it. This service is the same one that you see in the Control Panel. You'll even find the status flag the Windows NT displays in the Services dialog here. There are a couple of other values that could come in handy. For example, the GUIDs for the device drivers are here.

The third subkey under the SYSTEM key is Hardware Profiles. You'll find one or more sets of configuration keys under this key that are identified with numbers. Along with these configuration keys is a Current key that contains the current configuration. When you select a hardware profile in the Control Panel, what you're really doing is copying the contents of one of these numbered keys into the Current key — changing the configuration of your system. Each of the configuration keys contains the setup information for the hardware that you're using. For example, you'll find the resolution of your display adapter here. You won't find a complete hardware listing in this area — only the hardware with configuration settings appears in this section. For most people, that'll include their display adapter and not much more.

On to the fourth part of the System key — Services. This group of keys does just the opposite of what the Control key section does: It allows the user to change the way the system reacts. There are a few keys in this group that you should **never** touch. They include Arbitrators, Unknown, and Nodriver.

The majority of the Class key subkeys provide useful information. For example, the CDROM key can tell you about the drivers associated with the CD-ROM drive. In some cases, it might provide clues about why the CD-ROM drive fails to play music or interface with the system in some other way.

The Display key contains several subkeys related to the display adapter. This includes specifics about resolution and color combinations that the driver supports. (In one case, I was actually able to change this entry and provide 256-color support at 1024(768 resolution on one machine even though the driver originally showed that it didn't support this combination.) Of course, as with any other keys in this group, editing the values for these keys can result in a boot failure or other unforeseen occurrences.

HKEY_USERS

The HKEY_USERS hive contains a static listing of all the users of this particular registry file. It never pays to edit any of the information you find in this hive. However, you can use this hive for reference purposes. The reason for this hands-off policy is simple: None of the entries here will take effect until the next time the user logs into Windows NT, so you really don't know what effect they'll have until you reboot the machine. In addition, changing the settings for the current user is a waste of time since Windows NT will overwrite the new data with the data contained in HKEY_CURRENT_USER during log out or shutdown.

HKEY_CURRENT_CONFIG

The HKEY_CURRENT_CONFIG hive is the simplest part of the registry. It contains three major keys: CurrentDocInfo, Software, and System. Only two of these keys will be active on a desktop machine (Software and Settings). The CurrentDocInfo key is used exclusively with portable machines, and the exact content of this key will vary by machine.

Essentially, these entries are used by the GDI (graphics device interface) API (application programming interface) to configure the display and printer. You might find other devices here like scanners in some cases.

In most cases, the Software key contains two keys: Fonts and Microsoft. The Fonts key contains values that determine the base fonts for the system. The Microsoft key contains a variety of subkeys that determine things like the workstation's Internet settings.

The System key contains the current settings for the system from a GDI perspective. The three subkeys for the CurrentControlSet: Control, Enum, and Services contain settings and control information for the display adapter. Figure 5-11 shows the major keys in this hive.

Figure 5-11: The HKEY_CURRENT_CONFIG hive contains many GDI-related settings.

HKEY_DYN_DATA

The final hive, HKEY_DYN_DATA, is used exclusively for performance monitoring. You can't even view this hive using RegEdit because there really isn't a need to do so. In most cases, you won't need to do anything with the keys found in HKEY_DYN_DATA unless you're writing a custom counter. If you do decide to write your own counter, you'll need to modify the keys in this hive indirectly.

Deciding What to Store

Now that you have a better idea of what goes where in the registry, it's time to look at what kinds of things you might want to store there. The first thing we need to do is define the kinds of information that need to be stored. Essentially you can divide the whole question of what to store into two categories: data and settings.

Settings are application configuration information. A setting can be automatic. For example, if the user always opens a particular type of file from the list of files that your application supports, you could detect that and make the most common file type the default. Other kinds of automatic settings include configuration items like colors or fonts that the user changes from within a dialog box. You'd also group mandatory settings into this list like the acceptance of licensing terms that affect application operation, but don't require user input.

The user manually makes other settings, and you'll normally include a dialog asking if the user is sure they want to make them. For example, a user may want to add all of their contact information to a word processor so that output like envelopes and letters automatically contain the most current information. You wouldn't want to assume that changes to a dialog containing this kind of information are always correct, so you'd ask the user first before making the change in the registry. Obviously, any kind of change that could cause inadvertent data damage is another candidate for the manual change. You want to be certain that the user is aware of any risks before the change is made.

Data is an ambiguous designation for a wide variety of storage needs. Statistics is one category of data storage. For example, you might want to store the number of times that the user has created certain types of graphic elements and the sequence in which the elements were created. Storing this type of information for a graphics application could allow you to add a certain level of automation for the user. The application could actually anticipate what the user would do next.

Some data storage needs would almost qualify as settings. For example, a multi-monitor-capable application may need to store the number of the monitor on which the application was last used. In this way, the user could configure their desktop once and rely on automatic settings to produce the same results time after time.

Performance data is another potential storage need. There are situations in which you don't really need to write a full counter for performance monitoring, but it would be nice to provide the user with some application-usage statistics. Storing these figures within the registry would allow you to monitor performance without encumbering the system with useless services.

Objects are another potential storage need, but, at this point, you're starting to enter a gray area of registry usage. Sure, the registry can store binary data, but that binary data also remains in memory whether your application is in use or not. It's at this point that you have to start to consider another question, "How much registry storage is too much?"

Many people are starting to complain about the way that some applications bloat the registry unnecessarily. Unfortunately, the ease of using the registry, coupled with all of the security and reliability features it provides, makes it a good candidate for abuse. Of course, this means you should spend some time considering what data you want to store on disk in a separate file, which data goes with the data file itself, and what you want to place in the registry.

For the most part, you don't want to create huge registry entries that will make the storage problematic. Small values are best, even when working with binary data. In addition, it's a good idea to determine which entries fall into the gizmo area versus true-user aid. If you have a gizmo item that you want to store in the registry, it's usually wise to ask the user first.

Viewing the Registry

One of the first things you need to learn about when working with the registry is viewing its contents. You have to be able to find and read the data or settings you place in the registry before any entries you make will become useful. That's the purpose of this section of the chapter. We'll look at what you need to do to view the contents of the registry and to use the information you find as a basis for finding other information.

Our example program will allow the user to enter a file extension. The application will use that extension to search the registry for many of the entries we talked about earlier in this chapter. In other words, you'll see information about the commands, OLE 1, and OLE 2 entries that the registry has for that file extension. This example is fairly simple in concept, but it demonstrates many of the techniques you'll need to know as the chapter progresses.

CD-ROM Location: Chap05\ViewReg
Program Name: ViewReg.EXE
Source Code Modules in Text: ViewReg.RC and ViewRegDlg.CPP

Begin by creating a new MFC AppWizard (EXE) project named ViewReg (you saw how to perform this task in Chapter 3). Choose the Dialog-based option on the MFC AppWizard -Step 1 dialog. Use "Registry Viewer for File Extensions" as the title for your dialog on the MFC AppWizard - Step 2 dialog. At this point, the project settings are complete and you can click Finish. The New Project Information dialog you see should look like the one in Figure 5-12.

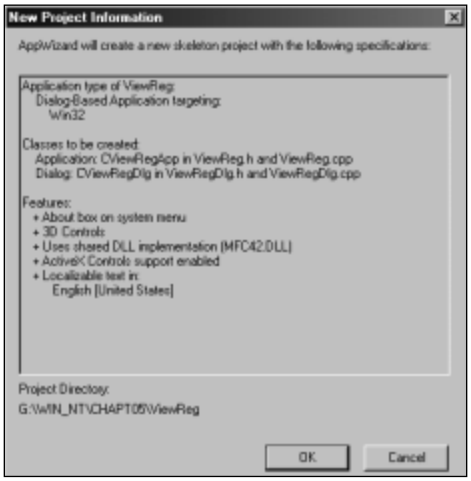

Figure 5-12: The New Project Information dialog for the ViewReg application

At this point, we need to add some controls to the IDD_VIEWREG_DIALOG dialog that the MFC AppWizard created for us. Figure 5-13 shows the arrangement of the dialog controls. Give the file extension edit control an ID of IDC_FILE_EXTENSION. The results edit control will have an ID of IDC_RESULTS. You'll also want to check the Read-only, Multiline, and the Auto VScroll options.

Creating some member variables comes next. Simply Ctrl+Double-click on each edit control in turn. The Member variable name field value for IDC_FILE_EXTENSION is m_fileExtension. You'll set this member variable up as a value, since we don't need to access the control itself and it's easier working with the CString that the value provides. The Member variable field value for IDC_RESULTS is m_results. You'll set this member variable up as a control since we need to actually change the value of the text within the control onscreen.

Now that we have some controls with associated member variables to work with, it's time to add some code. We'll override the usual functionality of the OK button by adding an event handler for it. Right-click the OK button and choose Events from the context menu. You'll see a New Windows Message and Event Handlers for class CViewRegDlg dialog like the one show in Figure 5-14. Highlight the BN_CLICKED message and click Add Handler. You'll see an Add Member Function dialog that asks what function name you want to use for the event handler. Accept the default

function name by clicking OK. You can now click the Edit Existing button to go to the newly added member function. Listing 5-1 shows the code that you'll need to add for this example.

Figure 5-13: The ViewReg dialog contains some simple controls for accepting input and displaying output.

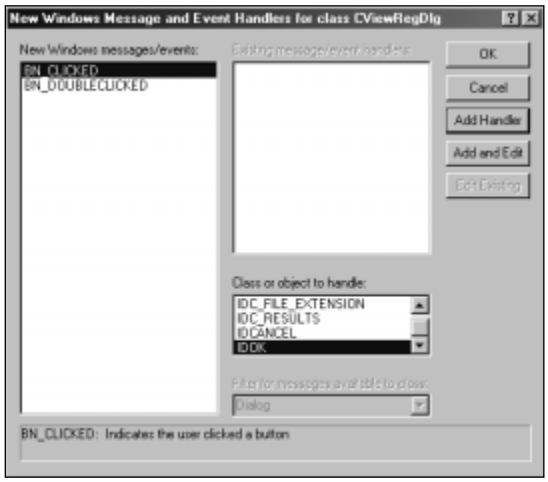

Figure 5-14: You can easily add message and event handlers for any control using this dialog.

Listing 5-1: **The OnOK Event Handler Source Code**

```
void CViewRegDlg::OnOK()
{
    CString    oKey;        // File extension key
    HKEY    hKey;        // File extension key handle.
    CString    oAssoc;        // File association for extension.
    HKEY    hAssoc;        // File association key handle.
    CString    oClsID;        // Class ID for extension.
    HKEY    hClsID;        // Class ID key handle.
    CString    oFType;        // Type of file.
    DWORD    dwType;        // Type of registry value.
    DWORD    dwSize;        // Size of the returned value.
    long    ulResult;    // Result of the operation.
    LPVOID    lpMessage;    // Error message buffer.
    CString    oDisp;        // Final results of our search.

    // Get the file extension we want to look for.
    m_fileExt.GetWindowText(oKey);

    // Try to open the file extension key.
    if (!RegOpenKeyEx(HKEY_CLASSES_ROOT,
        oKey,
        0,
        KEY_ALL_ACCESS,
        &hKey) == ERROR_SUCCESS)
    {
        // If we can't open the key, display an error message
        // and exit.
        AfxMessageBox("File Extension Key Not Opened", MB_OK);
        return;
    }

    // Query the default value of the file extension key,
    // which contains the value of the file association key.
    dwSize = 128;
    ulResult = RegQueryValueEx(hKey,
        "",
        NULL,
        &dwType,
        (PBYTE) oAssoc.GetBuffer(128),
        &dwSize);

    // Make sure you release the buffer once the file
    // association key is retrieved.
    oAssoc.ReleaseBuffer(-1);

    // Check for errors.
    if (!ulResult == ERROR_SUCCESS)
    {
        // If there was an error,
```

Continued

Listing 5-1 *(continued)*

```
    //create a text message for it.
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        ulResult,
        0,
        (LPTSTR) &lpMessage,
        0,
        NULL);

    // Display the error message.
    AfxMessageBox((LPCTSTR) lpMessage, MB_OK);

    // Free the memory used by the error message,
    // and then exit.
    LocalFree(lpMessage);
    return;
}

// Format the results.
oDisp = "File Association: " + oAssoc;

// Get the CLSID.
oAssoc = oAssoc + "\\CLSID";
if (!RegOpenKeyEx(HKEY_CLASSES_ROOT,
    oAssoc,
    0,
    KEY_ALL_ACCESS,
    &hAssoc) == ERROR_SUCCESS)
{
    // If we can't open the key, display an error message
    // and exit.
    AfxMessageBox("File Association Key Not Opened", MB_OK);
    return;
}

// Query the default value of the file association key,
// which contains the value of the class ID key.
dwSize = 128;
ulResult = RegQueryValueEx(hAssoc,
    "",
    NULL,
    &dwType,
    (PBYTE) oClsID.GetBuffer(128),
    &dwSize);

// Make sure you relese the buffer once the class ID
// key is retrieved.
oClsID.ReleaseBuffer(-1);
```

```
// Check for errors.
if (!ulResult == ERROR_SUCCESS)
{
    // If there was an error,
    //create a text message for it.
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        ulResult,
        0,
        (LPTSTR) &lpMessage,
        0,
        NULL);

    // Display the error message.
    AfxMessageBox((LPCTSTR) lpMessage, MB_OK);

    // Free the memory used by the error message,
    // and then exit.
    LocalFree(lpMessage);
    return;
}

// Format the results.
oDisp = oDisp + "\r\nClass ID: " + oClsID;

// Try to open the class ID key.
oClsID = "CLSID\\" + oClsID;
if (!RegOpenKeyEx(HKEY_CLASSES_ROOT,
    oClsID,
    0,
    KEY_ALL_ACCESS,
    &hClsID) == ERROR_SUCCESS)
{
    // If we can't open the key, display an error message
    // and exit.
    AfxMessageBox("Class ID Key Not Opened", MB_OK);
    return;
}

// Query the default value of the class ID key, which
// contains the type of file we're working with.
dwSize = 128;
ulResult = RegQueryValueEx(hClsID,
    "",
    NULL,
    &dwType,
    (PBYTE) oFType.GetBuffer(128),
    &dwSize);

// Make sure you relese the buffer once the file
```

Continued

Listing 5-1 *(continued)*

```
    // type value is retrieved.
    oFType.ReleaseBuffer(-1);

    // Check for errors.
    if (!ulResult == ERROR_SUCCESS)
    {
        // If there was an error,
        //create a text message for it.
        FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_SYSTEM |
            FORMAT_MESSAGE_IGNORE_INSERTS,
            NULL,
            ulResult,
            0,
            (LPTSTR) &lpMessage,
            0,
            NULL);

        // Display the error message.
        AfxMessageBox((LPCTSTR) lpMessage, MB_OK);

        // Free the memory used by the error message,
        // and then exit.
        LocalFree(lpMessage);
        return;
    }

    // Format the results.
    oDisp = oDisp + "\r\nFile Type: " + oFType;

    // Display the results of the search.
    m_results.SetWindowText(oDisp);

    // Close the open registry keys.
    RegCloseKey(hKey);
    RegCloseKey(hAssoc);
    RegCloseKey(hClsID);

    // Don't perform the default action!
    // CDialog::OnOK();
}
```

Gathering information from the registry doesn't have to be complicated, but it can be repetitive. In this example, we're performing the same sequence of events three times before producing a final result that lists the file association, class ID, and a file type for the file extension. However, it's important to realize that one set of events relies on the set of events before it. Our example will need to gather a specific kind of information at each step to trace through the registry. This information can be summarized like this:

1. Get the file association by looking at the default value of the file extension key.
2. Use the file association value to create a key that will be used to look up the class ID value.
3. Use the class ID value to create a key that can be used to look up OLE 2 information about the extension. One of these pieces of information is the file type, which usually has something to do with the name of the application used to open files with the file extension in question.

Let's go through an iteration of this sequence of events. The first thing we need to do is open the registry key using the RegOpenKeyEx( ) function. This function takes five arguments. The first and second arguments are the ones most likely to change every time you open a key. The first argument is a handle to an open key. You can either use a predefined handle like HKEY_CLASSES_ROOT, or the result of a previous RegOpenKeyEx( ) call. The second argument is the key you want to open. This example begins with the file extension you want to open and works through the list of keys from that point on. What you'll get, if you're successful, is a handle to the newly opened key as the last argument.

The registry-specific functions provided by the Windows API all return ERROR_SUCCESS if successful. This makes error handling easy because you always have a quick method for checking the success of the function call.

This example shows two error trapping methods. The first method is to check for ERROR_SUCCESS as part of the function call. That's the method shown here. If the RegOpenKeyEx( ) function doesn't return ERROR_SUCCESS, then we display a custom error message stating which registry key failed to open.

Now that we have a handle to the open registry key, we can query the key for a specific value (or the default value if no specific value is requested) using the RegQueryValueEx( ) function. This function takes the handle we opened as the first value and the name of the value you want to query as the second. Sending an empty string tells the function to query the default value of the key. The last three arguments contain the value type, the contents of the value, and the size of the value.

You can call RegQueryValueEx() without a return variable argument to determine the type of the variable you need to provide. The fourth RegQueryValueEx() argument will always return the type of the value, which you can then test using a simple switch. Make sure you always typecast the variable used to hold the contents of the value as type PBYTE as shown in the example.

At this point, we have the contents of the value we queried. Notice that we had to use the GetBuffer( ) member of the CString class to get the value in the first place. This means we have to release the buffer now before working with the CString variable anymore.

This brings us to the second type of error trapping used in this example. You can place the return value of RegQueryValueEx( ) into a variable and test it against ERROR_SUCCESS. If there was were an error, you can use the FormatMessage( ) function to create a text equivalent of the error-message value. This message is very generic in nature, but using this technique you test for a wide variety of error return values. Notice that we've asked FormatMessage( ) to allocate the memory required for the message string for us using the FORMAT_MESSAGE_ALLOCATE_BUFFER flag. It's important to use LocalFree( ) later to release the memory used by the message string variable or you'll create a memory leak. Once you've formatted the message, you can display it in a standard message box.

After all of the information is collected, we can display the results onscreen. Notice that the last thing we do is close the three registry keys using RegCloseKey( ). In addition, since we aren't performing the default OK processing for this dialog box, we don't need to call CDialog::OnOK( ). In most cases, you do want to perform the default processing so that your method doesn't have to include code to handle every possible contingency. This is one of the exceptions.

It's time to try the application out. Try clicking OK without providing any kind of file extension in the File Extension to Look For field and you'll see the error message shown in Figure 5-15. This is one of the generic error messages produced using the second error-trapping method. As you can see, the generic message is probably too generic to provide any useful information to the user. There are many situations in which the generic method works better, but it's a good idea to test a variety of situations to see what kind of responses you'll get.

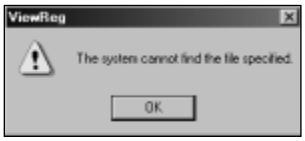

Figure 5-15: The FormatMessage() function produces this generic error message if you don't provide a file extension as input.

Try typing a non-existent file extension into the File Extension to Look For field and you'll see the File Extension Key Not Opened error message. As you can see, this error message at least provides a little information for the programmer, but not really enough to cure any obscure problems.

One of the more interesting file extensions to try is .TXT. If you're using a standard Windows NT setup, you should get a File Association Key Not Opened error message. In this case, a look in the registry explains everything. It turns out that the .TXT file extension doesn't have an OLE 2 entry — there's no CLSID key in the file association area of the HKEY_CLASSES_ROOT hive.

Most of you will have an application assigned to the .BMP file extension. Try typing this value into the File Extension to Look For field and click OK. You'll see a result similar to the one shown in Figure 5-16. Obviously, the results you get will vary from mine because the information you'll retrieve depends on the applications you have installed on the test machine. That's why a utility of this sort would come in so handy. You can easily extend it to create a diagnostic aid for file extension-related problems on a machine.

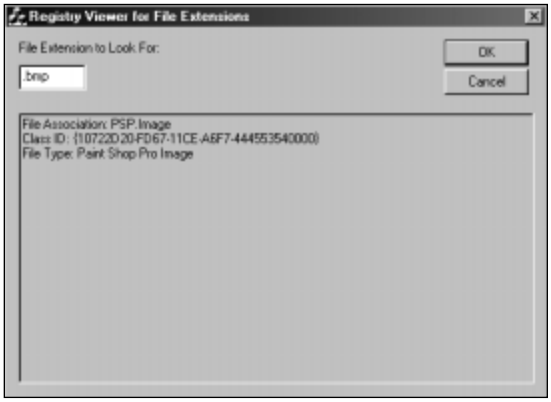

Figure 5-16: This utility could easily be extended to provide a wealth of information about the file associations on your machine.

Adding and Modifying Registry Entries

Adding and modifying registry entries are the two most common tasks that you'll perform. The first time an application is run or a user decides to set some application information, your application will add new registry entries. However, as time goes on the user may decide that a particular setting no longer suits their needs. At that point, your application will modify the registry entry. The example for this section will show you how to add and modify registry entries using a simple application configuration dialog.

This example modifies the registry, which is always risky. Make absolutely certain that you have a backup of your registry before you use the code in this section of the chapter. In addition, double-check your code before you compile and run the example. Even small spelling errors can have disastrous effects.

Needless to say, it doesn't pay to add registry entries if you can't read them. With this in mind, the example application will also read the registry entries and make appropriate changes to the application appearance as a result. This represents one of the more common ways that you'll read registry entries on a daily basis. Our example in the previous section was more of a utility use of registry reading that better showed how this process takes place.

CD-ROM Location: Chap05\AddReg
Program Name: AddReg.EXE
Source Code Modules in Text: AddReg.RC, MainFrm.CPP, MainFrm.H, and Options.CPP

You'll need to create a new MFC AppWizard (exe) project named AddReg. Choose the Single document option on the MFC AppWizard - Step 1 dialog. At this point, the project settings are complete and you can click Finish. The New Project Information dialog you see should look like the one in Figure 5-17.

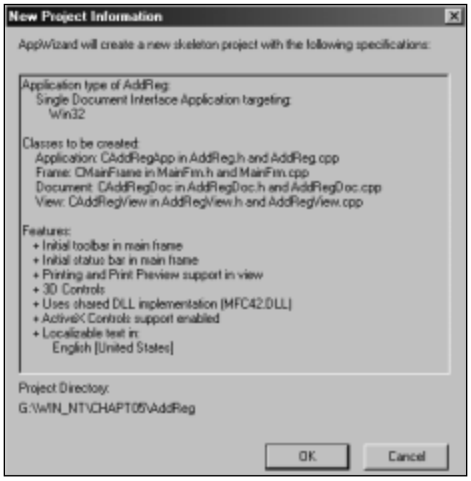
New Project Information
AppWizard will create a new skeleton project with the following specifications:
Application type of AddReg:
Single Document Interface Application targeting:
Win32
Classes to be created:
Application: CAddRegApp in AddReg.h and AddReg.cpp
Frame: CMainFrame in MainFrm.h and MainFrm.cpp
Document: CAddRegDoc in AddRegDoc.h and AddRegDoc.cpp
View: CAddRegView in AddRegView.h and AddRegView.cpp
Features:
+ Initial toolbar in main frame
+ Initial status bar in main frame
+ Printing and Print Preview support in view
+ 3D Controls
+ Uses shared DLL implementation (MFC42.DLL)
+ ActiveX Controls support enabled
+ Localizable text in:
English [United States]
Project Directory:
G:\WIN_NT\CHAPT05\AddReg
OK
Cancel

Figure 5-17: The New Project Information dialog for the AddReg application

Developing the Options Dialog

The first thing we'll need to add is a new menu entry. Figure 5-18 shows how the new menu entry should look. Give the Tools | Options... command an ID of ID_TOOLS_OPTIONS, a Caption of &Options..., and a Prompt of "Allows you to set your application options.\nApplication Options."

Figure 5-18: The new Tools | Options... command will allow the user to change the application's settings.

Now that we have a new menu entry, we need a configuration dialog to go with it. Right-click the Dialog folder on the ResourceView tab of the Workspace window and choose Insert Dialog from the context menu. You'll see a new dialog added to the resources. Visual C++ will automatically open the new dialog for you. Give the dialog a new ID by right-clicking the IDD_DIALOG1 entry, choosing Properties from the context menu, and typing IDD_OPTIONS in the ID field.

It's important to configure the Options dialog. The first thing you'll need to do is right-click on the dialog itself (in the design window) and choose Properties from the Context menu. Type Options in the Caption field, then close the Dialog Properties dialog.

We also need to add a few controls to the dialog as shown in Figure 5-19. These controls will affect overall application functionality in some way. As you can see, they'll make it possible for the user to turn certain application features on or off. (This dialog was kept simple because the overall goal of this example is to show how to add or modify registry entries.) Here's a list of captions and associated IDs for the controls.

ID	Caption
IDC_ALLOW_DOCK	Allow Toolbar Docking
IDC_ALLOW_SAVE	Allow File Save
IDC_ALLOW_PRINT	Allow Printing

Options dialogs are normally used to configure overall application performance, appearance, functionality, or user specific information such as name. Some applications also include a Customize dialog that allows the user to change the appearance of toolbars, menus, and other application interface features. It's important to keep the significance of the Options dialog in mind when designing your application. Sometimes it's better to include several menus and associated dialogs, rather than try to add every configuration need to the Options dialog.

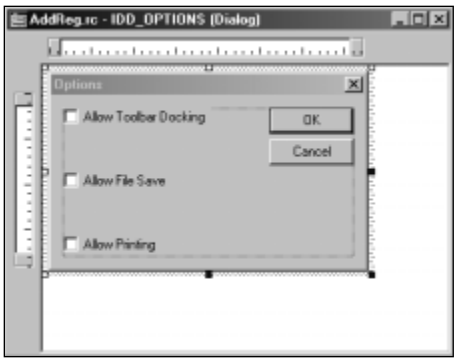

Figure 5-19: These checkboxes enable the user to change the sample application's functionality.

At this point, we need to create a class for the dialog so that we can display it onscreen and work with the various controls it contains. The following procedure will help you create the new dialog class. (This procedure is pretty generic — you can use it in modified form to create a class for just about any dialog resource.)

Try This

1. Ctrl+Double-click the Options dialog and you'll see an Adding a Class dialog like the one shown in Figure 5-20. This is where you decide whether to create a new class for the dialog, or attach the dialog to an existing class. In most cases, you'll want to create a new class for resources that you may need to access from more than one place in the application. Separate classes also come in handy when you think that you may need to use the same dialog in more than one project. Attaching the dialog to an existing class works fairly well if you only intend to access that class from one location in the application. For example, the About dialog is attached to the C<Application Name>App class (CAddRegApp class in this case) by default.

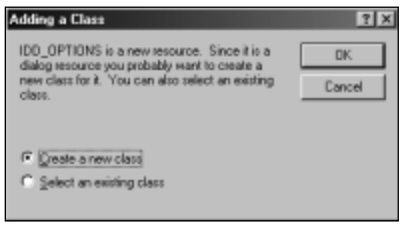
Adding a Class
IDD_OPTIONS is a new resource. Since it is a dialog resource you probably want to create a new class for it. You can also select an existing class.
OK
Cancel
Create a new class
Select an existing class

Figure 5-20: IDD_OPTIONS represents a new resource that requires a class of its own.

2. Choose the Create a New Class option, then click OK. You'll see the New Class dialog shown in Figure 5-21. This is where you'll configure the new class before asking Visual C++ to create it for you. Notice that the IDD_OPTIONS resource is already associated with the class. You need to create the association manually if you create the new class directly in MFC ClassWizard without Ctrl+Double-clicking the dialog resource first. This is also where you choose a base class for the dialog. In most cases, the CDialog class will work just fine, but it pays to be sure that there isn't a better solution for your particular application needs.

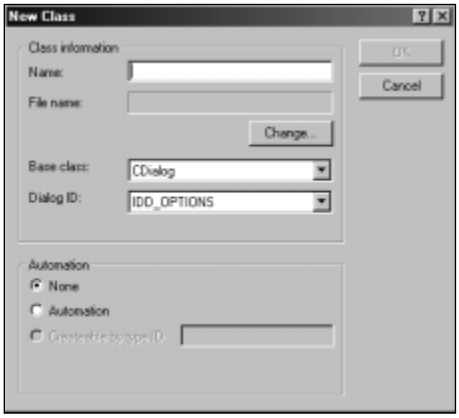

Figure 5-21: The New Class dialog box allows you to configure the new class before creating it.

3. Type **COptions** in the Name field. This will determine the class name for the new dialog.
4. Click OK. Visual C++ creates the new class for you.

Now that we have a class created for the Options dialog, we need to create member variables for each of the controls. Since we only need to know whether an item is checked, you'll want to choose Value in the Category field. The three checkboxes use Member variable name field values of: m_allowDocking, m_allowSaving, and m_allowPrinting, respectively.

Displaying the dialog is the last thing we'll cover in this section (the next subsection will show you how to process the user selections). First, you'll need to add a menu handler for the Tools | Options command to the CMainFrame class. We covered the procedure for doing this in Chapter 4. Listing 5-2 contains the code you'll need to add to the OnToolsOptions( ) method.

Listing 5-2: **The OnToolsOptions Menu Handler Source Code**

```
void CMainFrame::OnToolsOptions()
{
    // Display the Options dialog.
    dlgOptions.DoModal();
}
```

As you can see, the code is relatively simple. dlgOptions is an instance of the COptions class. We'll use the DoModal( ) method to display the resulting object. The code won't remain this simple; the following sections will show you how to modify this method (and others) so that we can process the user selections.

Notice that our code includes a reference to dlgOptions. We'll create this variable in the MainFrame.h file since we'll use it in more than one place in the example. The first thing you need to do to create this variable is to include the Options.H file at the beginning of the MainFrm.H file. Second, we'll need to create an instance of COptions. Listing 5-3 shows in bold the code you'll need to add.

Listing 5-3: **Adding an Instance of the COptions Class**

```
protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

    // Create an instance of the Options dialog.
    COptions    dlgOptions;
```

Default Setting Storage

Before we go any further, we need to look at a feature that has been in the SDI application all along because the MFC AppWizard added it for you. The most recently used (MRU) file list has to be stored somewhere in the registry. Otherwise, the application wouldn't know what files you had used from session to session and the MRU feature would be less than helpful. Listing 5-4 shows some MFC AppWizard-generated code taken from the CAddRegApp( ) method in the CAddRegApp class. Notice that the sample code defines both a default settings area (that you should change to reflect the name of your company or some other storage criteria) and loads the default settings for you (only the MRU in this case).

Listing 5-4: **Default Registry Entry Code for an SDI Application**

```
// Change the registry key under which our settings are stored.
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization.
SetRegistryKey(_T("Local AppWizard-Generated Applications"));

LoadStdProfileSettings();  // Load standard INI file options (including MRU)
```

It shouldn't surprise you that this particular code is provided for a lot of the applications we've created in the book so far. If you look at the HKEY_CURRENT_USER | Software | Local AppWizard-Generated Applications key and its subkeys, you should see a hierarchy like the one shown in Figure 5-22. Notice that the AddReg application already includes two keys: one for standard software settings and another for the MRU list.

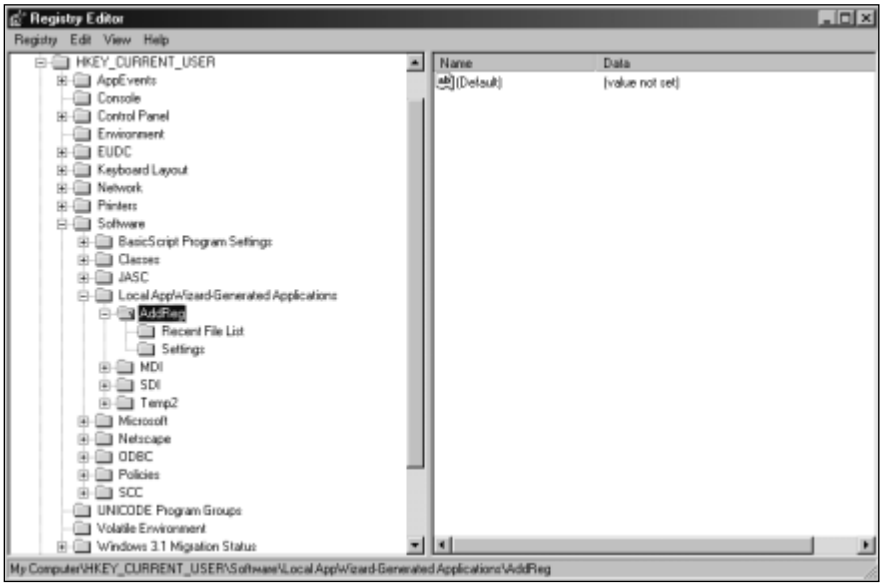

Figure 5-22: The MFC AppWizard creates some registry entries for you automatically as part of creating the application shell.

Both SDI and MDI applications automatically generate registry settings for you no matter what configuration options you choose in the MFC AppWizard. Other application types may generate registry settings for you depending on what settings you use when using the wizard. In addition, many of the non-application projects generate registry settings for you as a default. For example, the MFC AppWizard generates the required CLSID settings when you create an ActiveX control. Make sure you always run the component that you're creating one time before looking for any registry settings, since most MFC AppWizard-generated projects don't actually create the registry entries until you run the application for the first time.

Figure 5-22 should alert you to a couple of maintenance tasks that you need to be aware of during application development. The first is to change the code in the application initialization method so that the application settings get stored where you want them to be stored in the registry. The second is that you need to perform some clean-up of the registry after working with a test application. If you look at Figure 5-22 again, there's a Temp2 entry that got created while I was testing some applications features. Get enough of these extraneous entries and you could find yourself without sufficient registry space.

Since this is a sample application, you probably won't want to change the base code shown in Listing 5-4. However, this is one of those items that you will need to take care of during the application development cycle. We'll need to perform some processing of the registry entries that get added to this application, but that processing will take place in the CMainFrame class during the window initialization process. Instead of performing the default action, the registry entries will cause the application to perform an alternate setup. We'll visit this process in the next section of the chapter.

Processing the User Selections

It's time to create the code required to process user selections. The first task is to create a routine that looks at the users selections in the Options dialog. This is the place we'll make any new entries to the registry or modify the existing entries. Listing 5-5 shows the code you'll need to add to the OnToolsOptions( ) method found in the CMainFrame class (the additions are in bold).

Listing 5-5: **The Options Dialog OnToolsOptions() Event Handler Source Code**

```
void CMainFrame::OnToolsOptions()
{
  CWinApp*  pApp = AfxGetApp();  // Application pointer.

  // Ensure that our Options dialog reflects the current
  // registry settings.
  if (pApp->GetProfileInt("Settings", "Allow Docking", 0) == 0)
    dlgOptions.m_allowDocking = true;
```

```
  else
    dlgOptions.m_allowDocking = false;

  if (pApp->GetProfileInt("Settings", "Allow Saving", 0) == 0)
    dlgOptions.m_allowSaving = true;
  else
    dlgOptions.m_allowSaving = false;

  if (pApp->GetProfileInt("Settings", "Allow Printing", 0) == 0)
    dlgOptions.m_allowPrinting = true;
  else
    dlgOptions.m_allowPrinting = false;

  // Display the Options dialog.
  if (dlgOptions.DoModal() == IDOK)
  {
    // Make the appropriate registry changes.
    if (dlgOptions.m_allowDocking)
      pApp->WriteProfileInt("Settings", "Allow Docking", 0);
    else
      pApp->WriteProfileInt("Settings", "Allow Docking", 1);

    if (dlgOptions.m_allowSaving)
      pApp->WriteProfileInt("Settings", "Allow Saving", 0);
    else
      pApp->WriteProfileInt("Settings", "Allow Saving", 1);

    if (dlgOptions.m_allowPrinting)
      pApp->WriteProfileInt("Settings", "Allow Printing", 0);
    else
      pApp->WriteProfileInt("Settings", "Allow Printing", 1);

    // Setup the menu and toolbar.
    DoConfiguration();
  }
}
```

The first thing the code does is obtain a pointer to the application. It uses this pointer to get the current application registry settings using the GetProfileInt( ) function. The registry settings are used to set up the Options dialog before it gets displayed using the dlgOptions.DoModal( ) method that we talked about earlier.

There are profile calls for integer and string values. If you want to store a binary value, then you need to use the newer registry functions. Windows treats the profile calls differently depending on the operating system in use. However, unless you're using an older version of Windows, like Windows 3.*x*, the setting changes you make will appear in the registry, not an .INI file.

Once the user has viewed the dialog and made any required changes, the code determines whether the user clicked OK or cancel. If the user clicked OK, then the code makes any registry setting changes using the WriteProfileInt( ) function.

The DoConfiguration( ) function call implements the changes the user made on the Options dialog. I separated this code out because it needs to be called from two areas of the application (once after Options dialog changes and again during application initialization). You'll need to add an entry for this function in the Protected area of the MainFrm.H file, just like we did for the dlgOptions variable entry. Listing 5-6 contains the source code for the DoConfiguration( ) function.

Listing 5-6: **The DoConfiguration() Function Source Code**

```
void CMainFrame::DoConfiguration()
{
  CMenu*    pMenu = GetMenu();  // Application menu.
  CWinApp*  pApp = AfxGetApp();  // Application pointer.

  // Check to see if we're allowing the user to move
  // the toolbar around.  If not, disable the docking
  // capability.
  if (pApp->GetProfileInt("Settings", "Allow Docking", 0) == 0)
  {
    FloatControlBar(&m_wndToolBar,
      m_wndToolBar.GetBorders().TopLeft());
    m_wndToolBar.SetBarStyle(
      m_wndToolBar.GetBarStyle() | CBRS_GRIPPER);
    DockControlBar(&m_wndToolBar);
  }
  else
  {
    FloatControlBar(&m_wndToolBar,
      m_wndToolBar.GetBorders().TopLeft());
    m_wndToolBar.SetBarStyle(
      m_wndToolBar.GetBarStyle() ^ CBRS_GRIPPER);
    DockControlBar(&m_wndToolBar);
  }

  // Check to see if we're allowing the user to save.
  // If not, gray the corresponding menu items.
  if (pApp->GetProfileInt("Settings", "Allow Saving", 0) == 0)
  {
    pMenu->EnableMenuItem(ID_FILE_SAVE,
      MF_ENABLED);
    pMenu->EnableMenuItem(ID_FILE_SAVE_AS,
      MF_ENABLED);
  }
  else
```

```
  {
    pMenu->EnableMenuItem(ID_FILE_SAVE,
      MF_DISABLED | MF_GRAYED);
    pMenu->EnableMenuItem(ID_FILE_SAVE_AS,
      MF_DISABLED | MF_GRAYED);
  }

  // Check to see if we're allowing the user to print.
  // If not, gray the corresponding menu items.
  if (pApp->GetProfileInt("Settings", "Allow Printing", 0) == 0)
  {
    pMenu->EnableMenuItem(ID_FILE_PRINT,
      MF_ENABLED);
    pMenu->EnableMenuItem(ID_FILE_PRINT_PREVIEW,
      MF_ENABLED);
    pMenu->EnableMenuItem(ID_FILE_PRINT_SETUP,
      MF_ENABLED);
  }
  else
  {
    pMenu->EnableMenuItem(ID_FILE_PRINT,
      MF_DISABLED | MF_GRAYED);
    pMenu->EnableMenuItem(ID_FILE_PRINT_PREVIEW,
      MF_DISABLED | MF_GRAYED);
    pMenu->EnableMenuItem(ID_FILE_PRINT_SETUP,
      MF_DISABLED | MF_GRAYED);
  }

  // Redraw the menu so that the changes we've
  // made show up.
  DrawMenuBar();
}
```

The DoConfiguration( ) function has to perform three tasks, all of which are related to the three settings on the Options dialog. The first section of code sees if the Allow Docking option is checked. If it is, then the function gives the toolbar a gripper — the object used to dock and undock the toolbar. Notice that the code undocks the toolbar before it makes the change in toolbar appearance, then docks it again afterward. This technique ensures that the toolbar will actually get redrawn as needed to show the gripper. If the user unchecks the Allow Docking option, then the code simply removes the gripper.

The menu options all work the same. If the menu option is selected, then the associated menu entries are enabled. Note that contrary to Microsoft's documentation, it's best to use both the MF_DISABLED and MF_GRAYED flags when disabling the menu. You'll also want to use the DrawMenuBar( ) function to redraw the menu so any changes you make actually show up.

By default, Microsoft assumes that you want the application to automatically handle enabling and disabling the menu entries. What this means is that if you use the code shown in Listing 5-6 and don't disable the automatic updates, the code won't appear to do anything. Set the m_bAutoMenuEnable flag to false to allow any manual changes you make to the menu to appear.

The code that we've talked about, up to this point, is enough to ensure the user will see any changes they make to the Options dialog and that those changes get recorded in the registry. At this point, we need to add some code that will allow the application to read the registry at startup, and make any changes required to the application. After all, one of the major reasons to use registry settings is to ensure that user configuration settings are persistent from session to session. The code we need to change appears in the OnCreate( ) method of the CMainFrame class. Listing 5-7 shows you the code you need to add in bold (note that the MFC AppWizard generates much of this code for you).

Listing 5-7: **OnCreate() Method Changes Required to Implement Automatic Application Configuration**

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  HKEY  hKey;  // Handle of registry key we need to check.
  DWORD  dwType;  // Type of registry value.

  if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
    return -1;

  if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
CBRS_TOP
    | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
    !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
  {
    TRACE0("Failed to create toolbar\n");
    return -1;      // fail to create
  }

  if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
      sizeof(indicators)/sizeof(UINT)))
  {
    TRACE0("Failed to create status bar\n");
    return -1;      // fail to create
  }

  // TODO: Delete these three lines if you don't want the toolbar to
  //  be dockable
  m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
```

```
  EnableDocking(CBRS_ALIGN_ANY);
  DockControlBar(&m_wndToolBar);

  // Turn off automatic enabling of menu items.
  m_bAutoMenuEnable = FALSE;

  // See if the registry settings we need are in place.
    if (RegOpenKeyEx(HKEY_CURRENT_USER,
        "Software\\Local AppWizard-Generated Applications\\AddReg\\Settings",
        0,
        KEY_ALL_ACCESS,
        &hKey) == ERROR_SUCCESS)
    {
    // If so, then check for the required registry values.
    if (RegQueryValueEx(hKey,
      "Allow Docking",
      NULL,
      &dwType,
      NULL,
      NULL) == ERROR_SUCCESS)

      // If everything is in place, configure the application.
      DoConfiguration();
    }

  // Close the open registry keys.
  RegCloseKey(hKey);

  return 0;
}
```

Let's talk about the easy part of the code first. We need to set the m_bAutoMenuEnable flag to false to allow any manual changes to the menu to appear. This flag must be set before you attempt to perform any application configuration or some configuration may not take affect.

The registry specific code in Listing 5-7 should look pretty familiar. It's an adaptation of the code we used in the RegView application. The first thing we do is attempt to open the Settings registry key. If this key doesn't appear in the registry, then this is the first time the application is being run and any default settings are just fine. The next thing that the code does is attempt to query a specific setting value. Even if the user has opened the application quite a few times, they may not have made any configuration changes. If not, then the default settings will get used. If there are configuration settings, then the code calls DoConfiguration( ), which sets the application up to user specifications as described earlier. The last registry function is to close the registry key that we've opened.

At this point, you can compile and test the application. If you make a change in the Options dialog, you'll notice that it has an immediate effect on the application. In addition, when you close the application and restart it, you'll notice that the application automatically reconfigures itself as needed. Most important of all, if you look into RegEdit now, you'll see that there are some new values associated with the Settings key like the ones shown in Figure 5-23.

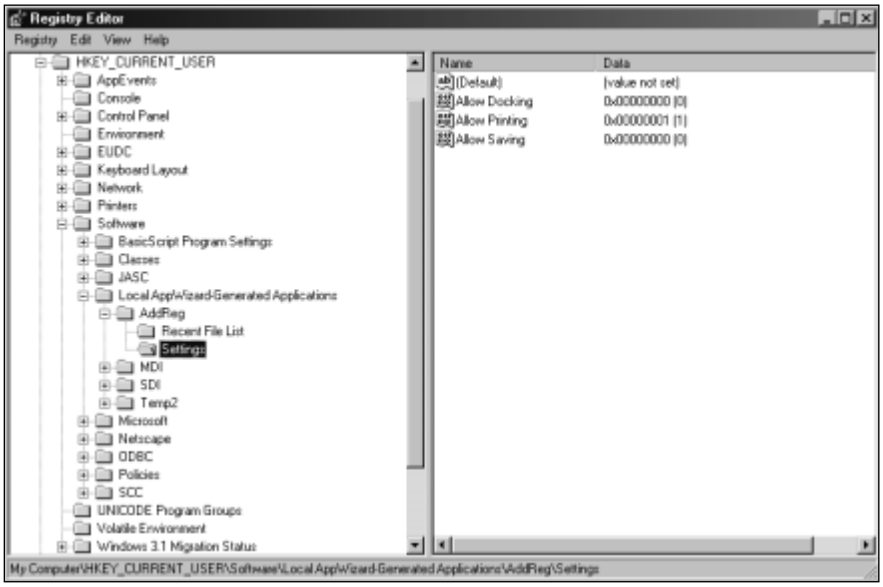

Figure 5-23: The code we've added to AddReg stores application settings in the registry

Removing Registry Entries

While the number of times you actually need to remove registry entries is relatively small, there are still times you need to do it. The most common example is when the user needs to uninstall the application. In other cases, you might want to provide a clear or default feature for your application that sets the registry back to its native state, which may include removing registry entries. This section will show you how to add a Tools | Clear Options command to the sample application we started in the previous section.

The first thing you'll need to do is add a Clear Options entry to the Tools menu and create an event handler for it. The Prompt for this entry is, "Clear any application options from the registry.\nClear Application Options" and the ID is ID_TOOLS_CLEAR_OPTIONS. Once you have the new menu entry in place, create an event handler for it (we covered this procedure in Chapter 4).

It's time to add some code for our new menu entry. Listing 5-8 shows the code you'll need to add to the CMainFrame class OnToolsClearOptions( ) method.

Listing 5-8: The OnToolsClearOptions() Event Handler Source Code

```
void CMainFrame::OnToolsClearOptions()
{
  CMenu*    pMenu = GetMenu();  // Application menu.
  CWinApp*  pApp = AfxGetApp();  // Application pointer.

  // Delete the application settings.
  RegDeleteKey(HKEY_CURRENT_USER,
    "Software\\Local AppWizard-Generated Applications\\AddReg\\Settings");

  // Reset the application settings to default.
  FloatControlBar(&m_wndToolBar,
    m_wndToolBar.GetBorders().TopLeft());
  m_wndToolBar.SetBarStyle(
    m_wndToolBar.GetBarStyle() | CBRS_GRIPPER);
  DockControlBar(&m_wndToolBar);
  pMenu->EnableMenuItem(ID_FILE_SAVE,
    MF_ENABLED);
  pMenu->EnableMenuItem(ID_FILE_SAVE_AS,
    MF_ENABLED);
  pMenu->EnableMenuItem(ID_FILE_PRINT,
    MF_ENABLED);
  pMenu->EnableMenuItem(ID_FILE_PRINT_PREVIEW,
    MF_ENABLED);
  pMenu->EnableMenuItem(ID_FILE_PRINT_SETUP,
    MF_ENABLED);

  // Clear the MRU list by killing the first file entry.
  pApp->AddToRecentFileList("");

  // Kill any MRU menu entries beyond the first one.
  pMenu->DeleteMenu(ID_FILE_MRU_FILE2, MF_BYCOMMAND);
  pMenu->DeleteMenu(ID_FILE_MRU_FILE3, MF_BYCOMMAND);
  pMenu->DeleteMenu(ID_FILE_MRU_FILE4, MF_BYCOMMAND);

  // Delete the most recently used file list.
  RegDeleteKey(HKEY_CURRENT_USER,
    "Software\\Local AppWizard-Generated Applications\\AddReg\\Recent File
List");

  // Make the changes permanent.
  RegFlushKey(HKEY_CURRENT_USER);
}
```

The registry API functions can behave differently in Windows 95/98 than they do in Windows NT and Windows 2000. For example, you can use a single call to delete all of the keys in a tree in Windows 95/98 by specifying the key at the root of the tree. In Windows NT and Windows 2000, you must delete the individual keys in the tree one at a time. While this may seem to be an arbitrary increase in the difficulty of using the registry, using the Windows 2000 technique actually ensures that only the desired keys get deleted—reducing the probability of accidental key deletions.

The registry key deletion code is relatively straightforward. All we need to provide is the predefined handle for the HKEY_CLASSES_USER hive and the path of the key that we want to delete. There are two keys to delete: the one for the application configuration (Settings) and the one for the most recently used file list (Recent File List).

Unfortunately, simply deleting the registry keys doesn't return the application to its default state. We've already looked at the code required to implement the modifications on the Options dialog. The only difference, in this case, is that we'll change the application back to its default state, rather than a user-specified state.

Setting the most recently used file list back to its normal state is a little trickier. The technique used in this case is to set the first file entry to a blank value. This value won't get recorded in the registry and also returns the first file entry in the menu list to the default of "Recent File." Changing the first menu entry won't get rid of the other menu entries that might be in use. Our sample application can store up to four recent files, so we've got to manually delete the remaining three entries.

Modifying Registry Security

The registry, as previously stated, is a hierarchical database of settings for various Windows objects including applications and components. In most cases, you want to provide any applications with a need to know about your application or component full access to the settings you make in the registry. It's not practical to force the administrator to add users every time someone new needs to access common Windows elements. In addition, using security would slow the application down because Windows would need to verify the identity of the person requesting access to a particular set of registry keys and values.

However, there are some situations where you don't necessarily want someone to have complete access to the registry. For example, you'll find that you can't open the HKEY_DYN_DATA hive using RegEdit. What you'll see, instead, is an error message saying that the hive is unavailable. The reason for this restriction is quite simple. There are other applications designed to interpret the contents of this set of registry keys so there really isn't a need to access them externally. In fact, accessing these keys externally could negatively impact the results of any performance-gathering mission that another user might conduct.

Obviously, you might also run into a situation where the contents of a set of registry keys are confidential. In fact, you'll find that many of the security keys in the registry are protected for this very reason. The owner of the key doesn't want security breached because someone has looked through the registry and made changes to certain application settings externally using RegEdit.

Since it's fairly rare to secure a registry key, Windows assumes that you don't want any security, no matter what key you create. The exception to this rule is when you create a subkey for a key that currently has security assigned. Any security assigned to the original key will also be assigned to the new key.

Windows provides two functions for getting and setting security: RegGetKeySecurity( ) and RegSetKeySecurity( ). Both functions deal with security descriptors, a topic we'll cover in Chapters 17 and 18 of the book. Other than the security descriptor, all you need to use these functions is a handle to the key (a topic we've discussed extensively in the chapter) and a description of the contents of the security descriptor.

Summary

In this chapter, we discussed some of the things you can do with the registry. It's important to realize that the registry is a complex database that contains many entries, some of which are automatically made for you. Application settings and configuration entries are just one small category of data that the registry contains. A study of the registry can even help you better understand how Windows works as a whole, since everything from components to major subsystems all rely on it. Here are some additional conclusions you can draw from the contents of this chapter.

- The registry is a database consisting of keys and values.
- Registry values can contain information in three basic formats: binary, string, or DWORD.
- The HKEY_CLASSES_ROOT hive contains most of the information required for file associations. It also contains essential COM information for both applications and components.
- The HKEY_CURRENT_USER hive contains all of the user-specific information for the user that is currently logged in. This information includes everything from application settings to user configurable elements of Windows.
- The HKEY_LOCAL_MACHINE hive contains all of the settings that affect the machine as a whole. In other words, any non-user-configurable settings get stored here, along with many of the machine's hardware settings.
- The HKEY_USERS hive is a repository of information for all of the users who have access to the machine in question. The contents of this hive get copied to HKEY_CURRENT_USER when a user logs in and back to this hive when the user logs out.

- The HKEY_CURRENT_CONFIG hive contains configuration-specific information that is mainly used by the GDI.
- The HKEY_DYN_DATA hive contains the settings for counters and other performance monitoring features of the system.
- Normally you'll use a combination of the RegOpenKeyEx( ) and RegQueryValueEx( ) functions to obtain the contents of a particular registry key.
- Error-trapping and reporting is extremely important when working with the registry. One of the most generic, yet complete, error-reporting methods is to use the FormatMessage( ) function.
- When working with applications, you can choose to use the GetProfileXXX and WriteProfileXXX functions to read and write application-specific registry entries.
- You'll use the RegDeleteKey( ) and RegDeleteValue( ) functions to remove keys and values from the registry.
- It's important to modify the user interface and functionality of an application to match any registry changes you make that affect these features.
- All registry keys start out with little or no security unless that security was previously assigned by Windows or another application.

✦ ✦ ✦